Universidad
Carlos III de Madrid
www.uc3m.es

Universidad
Carlos III de Madrid
www.uc3m.es

Grado en ingeniería electrónica industrial y automática.

Curso académico 2014-2015

Trabajo Fin de Grado

Aplicación Android para Identificación de Señales de Tráfico.

Mario Morante Díaz

Tutor/es:

Juan Carmona Fernández

05/10/2015; Escuela Politécnica de Leganés.

Universidad Carlos III de Madrid
www.uc3m.es

Agradecimientos.

Quisiera agradecer a mis padres todo el apoyo, la motivación y la ayuda que me han dado a lo largo de todos mis estudios y principalmente de mi grado en ingeniería electrónica industrial y automática, sin ellos no habría sido posible llegar hasta aquí.

Agradecer también a mis amigos por proporcionarme un entorno de responsabilidad, ayudarme en los momentos difíciles y en las duras y largas jornadas de biblioteca en épocas de exámenes.

Por último hacer mención a mi hermana mis abuelos y el resto de familia, en especial a mi abuelo que siendo ingeniero industrial superior he encontrado en él un ejemplo a seguir y el cual se ha preocupado de mi carrera desde el primer momento.

RESUMEN

El objetivo principal de éste Trabajo de Fin de Grado es la creación de una aplicación que con el software Eclipse y la ayuda de la biblioteca OpenCV sea capaz de leer y reconocer señales de tráfico en un terminal con sistema operativo Android.

Estas señales de tráfico son las que podemos encontrar en las carreteras españolas, la única restricción que se encuentra es que deben estar en nuestra base de señales para que puedan ser reconocidas.

La aplicación es desarrollada para el Laboratorio de Sistemas Inteligentes (LSI) de la Universidad Carlos III de Madrid (UC3M), y ésta aplicación forma parte de una aplicación base ya desarrollada por dicho departamento, la cual tiene otras aplicaciones ya incorporadas.

Esto ha sido posible gracias a la combinación de varias herramientas de software y al posterior desarrollo del código en lenguaje java.

Hoy en día la visión por computador es una disciplina que está en auge ya que con el avance de las tecnologías y de la electrónica se está investigando y desarrollando infinidad de proyectos y aplicaciones que facilitan cualquier tipo de tarea de nuestra vida cotidiana.

Por lo tanto la principal finalidad de este Trabajo de Fin de Grado es la de facilitar y ayudar a la investigación y desarrollo de una conducción autónoma, acercando al usuario de calle a este mundo gracias al desarrollo de ésta aplicación, con la cual el usuario puede ver y comprobar que realmente si se puede llegar a conseguir leer señales e interpretarlas con un Smartphone, tablet o cualquier elemento con Android, siempre y cuando disponga de cámara.

ABSTRACT

The main objective of this EOG Project is the creation of an app which with the help of Eclipse and OpenCV software should be able to read and recognize traffic signals with an Android device.

All these traffic signals are those which we can find in the Spanish roads, the only restriction is that all this signals have to stay in our database in order to be recognized.

The app is developed for the Intelligent System Laboratory of the Carlos III University of Madrid; this app is a part of a base-app already designed by this department, which has also other apps incorporated.

All this have been possible thanks to the combination of several software tools and the java code design.

Nowadays the computer vision is a booming discipline because with the advance of technology and electronics are researching and developing a lot of Works and apps that makes easier any kind of our daily tasks.

In conclusion the end purpose of this EOG Project is to make easier and help the investigation and development of a handless car drive, approaching the citizen to this world thanks to the development of this app, with which the user can see and check by his self that it is really possible read and interpret traffic signals with a Smartphone, tablet or any other Android device, as long as it has a camera incorporated.

Universidad Carlos III de Madrid
www.uc3m.es

ÍNDICE GENERAL

ÍNDICE DE FIGURAS.

1. Introducción.

El sistema de visión por computador o también conocido como visión artificial es un campo que está siendo desarrollado desde el inicio de la mejora del procesado de imágenes desde comienzos de los años veinte.

Está enfocado a la extracción de características e información a partir de imágenes con el fin de poder ofrecer soluciones a problemas cotidianos del mundo real.

La seguridad vial está presente hoy en día en nuestro país en el que los ciudadanos han de concienciarse de su importancia ya que consiste en la prevención tanto de accidentes como los efectos de éstos, al mismo tiempo que se refiere a las tecnologías empleadas para dicho fin, es aquí en donde entra en juego el campo de la visión por computador, que sirve de gran ayuda para complementar la información obtenida relacionada con este tipo de seguridad.

1.2 Objetivo.

El objetivo del desarrollo de este trabajo de fin de grado es el de la creación de una aplicación Android para la detección de diferentes señales de tráfico que podemos encontrar en las vías de circulación españolas.

El proyecto desarrollado en este TFG en sí consiste en adaptar a una aplicación base ya existente, perteneciente al Laboratorio de Sistemas Inteligentes (LSI) en la universidad Carlos III de Madrid, una subaplicación de la misma para poder implementar las funciones previamente mencionadas de detección de señales haciendo uso al mismo tiempo de una biblioteca libre de visión artificial llamada OpenCV.

Inicialmente el proyecto nace con una idea de reconocimiento de señales basada en la ya existente en los coches autónomos, pero implementada y adaptada a nivel usuario, es decir en un Smartphone con sistema operativo Android que cualquier ciudadano puede poseer.

El hecho de haberlo implementado en esta plataforma y no en otra como iOS o Windows Phone es básicamente la política de software libre y gratuito que tiene la plataforma Android, lo que hace que este mucho más desarrollado y avanzado en este sistema operativo frente a los otros.

La aplicación base posee diferentes proyectos ya implementados y probados con resultados bastante satisfactorios y favorables a los que yo he añadido mi proyecto o aplicación que hace que en conjunto sea una aplicación que con las futuras mejoras e ideas a desarrollar sea una aplicación muy completa que relaciona la seguridad vial y la visión por computador en un mismo punto.

2. Estado del arte.

2.1 Sistema operativo Android.

Android es un sistema operativo diseñado inicialmente para dispositivos móviles con pantalla táctil, ya sea Smartphone o Tablet. Se basa en el núcleo Linux y hoy en día se ha añadido algún nuevo dispositivo con Android debido al avance tecnológico que no cesa, se trata de los Smart Watches o relojes inteligentes; los Smart TV o televisores inteligentes; y también se ha conseguido adaptar este sistema operativo a los automóviles.

Comenzó respaldado por Google ya que inicialmente era una empresa independiente llamada Android Inc., y posteriormente Google realizó la compra de la misma en 2005. Comenzó a formar parte en 2007 de una fundación llamada Open Handset Alliance (un consorcio de compañías de hardware, software y telecomunicaciones) con el fin de poder avanzar con los estándares abiertos de los dispositivos móviles.

Es entonces cuando aparece en 2008 el primer dispositivo móvil con sistema operativo Android, se trata del HTC Dream. Fue tal el auge de ventas de los dispositivos Android que se ha convertido en los dispositivos más vendidos, incapaces de superarles sus dos principales competidores juntos (Windows Phone e iOS). [1]

En el siguiente gráfico se puede ver la presencia en el mercado global de los distintos sistemas operativos que existe, en donde se aprecia la absoluta mayoría del sistema Android frente a otros:

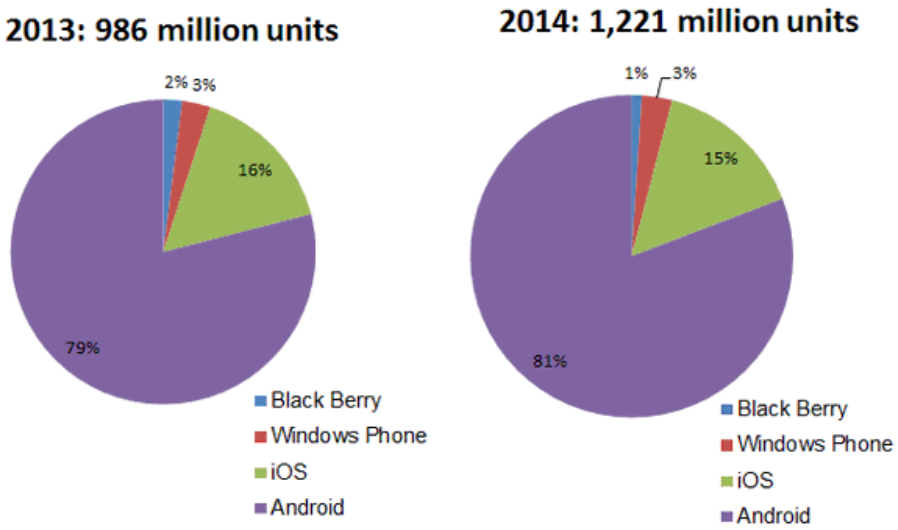

Figura 1: Sistemas operativos existentes, en Uds.

Éste éxito ha dado lugar a una guerra de patentes de Smartphones entre algunas empresas tecnológicas, y además entre el 2013 y 2014 se filtraron documentos en los que se identificaba el sistema operativo como objetivo de agencias de inteligencia internacionales.

En cuanto al desarrollo de las versiones de Android se presentó una versión inicial llamada Android Open Source Project (AOSP). Posteriormente el 25 de junio de 2014 en la Conferencia de Desarrolladores Google I/O Google mostró una evolución de la marca para ampliar mercados unificando hardware y software. Fue en este momento en el que presentaron productos nuevos como Android TV, Android Auto, Android Wear o una serie de "Smartphone" de baja gama bajo el nombre de Android One. Todo esto sirvió para estabilizar su imagen frente al público y los mercados.

Todas las aplicaciones hoy en día se pueden encontrar en Google Play Store. Anteriormente llamado Android Market. Ésta es la tienda online que Google ha desarrollado para Android. Gracias a esta aplicación el usuario puede navegar y descargar todas aquellas aplicaciones publicadas y desarrolladas por los desarrolladores, retribuyendo así de esta forma un 70 % a los desarrolladores, ingresando Google el porcentaje restante.

Los usuarios también tienen otras alternativas para instalar aplicaciones como puede ser la tienda de aplicaciones de Amazon llamada Amazon Appstore, otra llamada SlideME, o directamente en el caso de tener la APK de la aplicación.

En cuanto a seguridad del terminal, Symantec en 2013 desarrolló un estudio en el que demostraba que en comparación con iOS Android resulta ser un sistema menos vulnerable. El estudio habla de 13 vulnerabilidades graves para Android y 387 vulnerabilidades graves para iOS, al mismo tiempo que habla de los ataques en ambas plataformas, en este caso Android se queda con 113 ataques nuevos en 2012 a diferencia de iOS que se queda en 1 solo ataque. Esto es algo que hace que tanto Google como Apple hagan cada vez más esfuerzo en hacer sus sistemas operativos más seguros incorporando más seguridad tanto en sus sistemas operativos como en sus mercados oficiales. [2]

Figura 2: Logotipo de Android.

2.2 OpenCV.

OpenCV (Open Source Computer Vision) es una biblioteca libre de visión artificial que Intel presentó en enero del 1999. Fue en esta fecha en la que se presentó la primera versión y desde entonces ha servido como biblioteca para desarrollar infinidad de aplicaciones. Estas aplicaciones van desde sistemas de seguridad con detección de movimiento, hasta aplicativos de control de procesos donde se requiere reconocimiento de objetos. Esto es gracias al tipo de licencia que tiene, que es la licencia BSD gracias a la cual puede ser usada libremente para propósitos comerciales y de investigación teniendo en cuenta siempre las condiciones que OpenCV expresa.

Figura 3: Logotipo de OpenCV.

Se trata de una multiplataforma, existen versiones para los siguientes sistemas operativos: GNU/Linux, Mac OS X y Windows. Cuenta con un número de funciones superior a 500 que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos entre los que destaca el reconocimiento facial, visión estérea, calibración de cámaras y visión robótica. [3]

OpenCV tiene varios interfaces de programación como C++, Python y java y soporta Windows, Linux, Mac OS, iOS and Android. OpenCV fue diseñado para mejorar la eficiencia computacional enfocada fuertemente en las aplicaciones de tiempo real. Esta optimizado para lenguaje C/C++ y puede funcionar mejor con un procesamiento multi-core.

Está adaptado para todo el mundo y tiene más de 47 mil usuarios y el número de descargas excede el de los 9 millones. [4]

Tiene más de 2500 algoritmos optimizados en la biblioteca y las descargas ascienden cerca de 200 mil al mes. [5]

Es utilizado en especial por compañías, grupos de investigación y cuerpos gubernamentales. Algunas de las compañías que utilizan OpenCV son Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, y Toyota. También hay muchas startups como Applied Minds, VideoSurf, y Zeitera que utilizan también esta biblioteca.

Otros ejemplos de usos de OpenCV son la toma de imágenes conjuntas de Streetview, detección de intrusiones en video vigilancia en Israel, seguimiento de equipamiento minero en

China, en donde además se utiliza también en el filtro “Green Dam” de internet de este mismo país. Willow Garage utiliza robots con esta herramienta para navegar y coger objetos, además en Europa se utiliza para la detección de ahogamientos en piscinas, en España y Nueva York se utiliza para la ejecución de arte interactivo, inspección de etiquetado de los productos de las fábricas de todo el mundo, y la detección rápida de rostros en Japón.

En 2012 se creó la fundación de OpenCV con una nueva interfaz y una nueva estructura. Ahora abarca más de 40 "constructores" diferentes, que prueba OpenCV en varias configuraciones en diferentes sistemas operativos tanto para móvil como para terminales (un "constructor" es un entorno utilizado para construir la biblioteca bajo una configuración específica. El objetivo es comprobar que la biblioteca se construye correctamente en la configuración especificada).

OpenCV4Android es el nombre oficial del puerto Android de la librería OpenCV. OpenCV comenzó a soportar Android con una versión “alpha” en el año 2010 con la versión OpenCV 2.2. Fue entonces cuando NVIDIA entró en el proyecto acelerando el desarrollo de OpenCV para Android con el desarrollo de la versión 2.3.1 beta. Esta versión inicial incluía una API OpenCV de Java y el soporte nativo de cámara. La primera versión oficial no-beta apareció en Abril del año 2012 con el OpenCV 2.4.

OpenCV4Android admite dos idiomas para codificar aplicaciones OpenCV para funcionar en dispositivos basados en Android, la forma más fácil de desarrollar el código es el uso de la API de Java OpenCV.

OpenCV expone la mayoría (pero no todas) de sus funciones a Java. Esta aplicación incompleta puede plantear problemas si necesita una función de OpenCV que todavía no ha recibido el apoyo de Java. Antes de la elección de utilizar el lenguaje Java para un proyecto de OpenCV, hay que revisar la API de Java para funciones OpenCV que su proyecto requiera.

A continuación se muestra una representación esquemática de algunas de las operaciones que se pueden llevar a cabo con OpenCV4Android. [6]

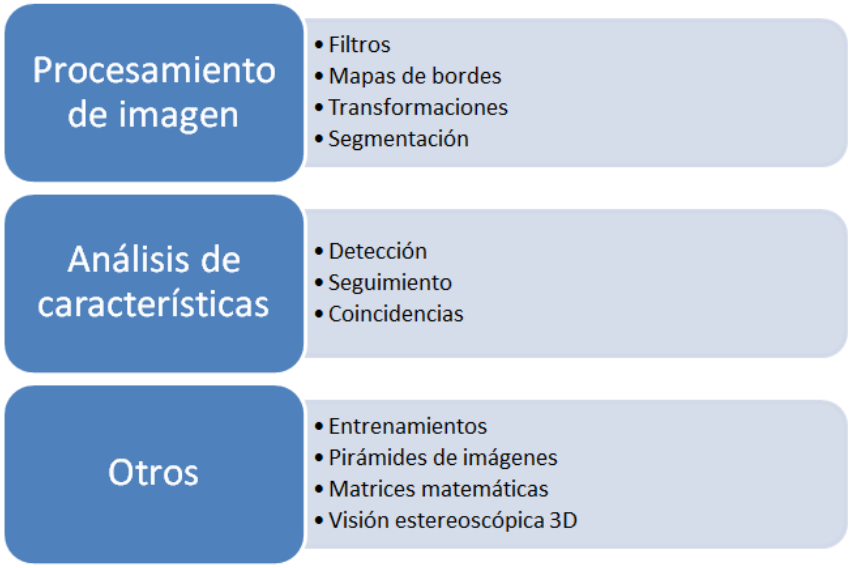

Figura 3: Operaciones con OpenCV.

Un factor a tener en cuenta es que para poder utilizar la biblioteca de OpenCV en el dispositivo móvil es necesario tener descargado desde la tienda de aplicaciones "Google Play" la aplicación *OpenCV Manager*.

2.3 Evolución de la visión por computador en vehículos.

Hoy en día el medio de transporte más utilizado es el automóvil. Es un transporte relativamente masificado lo que da lugar a problemas como el tráfico excesivo, gran número de accidentes que año tras año se intentan reducir y poco a poco se consigue gracias a los avances y a la seguridad vial, y por último otro asunto es la contaminación que aumenta consideradamente. Hablando de números se puede afirmar que el número de vehículos ha aumentado al doble en una franja de 15 años desde el año 1985 hasta el año 2000 [7], existiendo en la actualidad 31,04 millones de coches en circulación frente a 21,3 millones en el año 1998 lo que equivale a un incremento de un 45,7.[8]

Como consecuencias de este considerable aumento, la más importante es la congestión de vehículos debido a la masificación de vehículos en las infraestructuras, por lo tanto uno de los objetivos de los Sistemas Inteligentes de Transporte (SIT) es tratar de mejorar las infraestructuras, incrementar la seguridad y todo ello se lleva a cabo a través de las tecnologías de la información. [9]

Los Sistemas Inteligentes de Transporte tienen como objetivo tratar un conjunto de soluciones tecnológicas de las telecomunicaciones y la informática diseñadas para mejorar la operación y seguridad del transporte terrestre, tanto para carreteras urbanas y rurales, como

para ferrocarriles. Este conjunto de soluciones telemáticas también pueden utilizarse en otros modos de transporte, pero su principal desarrollo ha sido orientado al transporte terrestre.

Algunas de las aplicaciones de los SIT son:

- Cobro electrónico de peajes.
- Peajes convencionales.
- Tarifas de congestión.
- Vigilancia automática de infracciones.
- Sistema de notificación de emergencia a bordo del vehículo. [10]

El SIT se centra en dos campos para poder mejorar, uno de ellos supondría mejorar y actualizar las infraestructuras, que pensando en las infraestructuras totales de todo un país, esto es algo imposible económicamente hablando. Por otra parte se centra en la mejora e introducción de nuevas capacidades en los vehículos. Esto va enfocado a una conducción automática.

Esta conducción automática necesita ser trabajada pensando en diferentes aspectos:

• Seguimiento del borde de la carretera.

• Mantenimiento de la distancia de seguridad.

• Regulación de la velocidad dependiendo del estado del tráfico y del tipo de la carretera.

• Adelantamientos de otros vehículos.

• Trazado automático de la ruta más corta.

• Movimiento y aparcamiento dentro de las ciudades.

Existen varias dificultades a la hora de implementar una conducción automática:

• Técnicas. Los algoritmos deben ser rápidos ya que trabajan en tiempo real y no permitirse ningún fallo, es decir fiabilidad del 100%.

• Económicas. No debe suponer un aumento del coste del vehículo.

• Psicológicas. Los pasajeros deben confiar plenamente en el sistema ya que no son ellos los que tienen el control.

• Legales. Si el sistema falla desemboca en un accidente por lo tanto sería difícil encontrar responsables.

Debido a todos estos problemas es bastante dificultoso implementar una conducción automática de golpe, por lo que se está centrando en la ayuda a la conducción, que probablemente sea un paso intermedio para dar el gran salto al coche autónomo, al mismo tiempo que sirve para mejorar y perfeccionar todas las técnicas de conducción automática.

Esta ayuda a la conducción se desarrolla mediante los Sistemas de Ayuda a la Conducción, y los más importantes son los siguientes:

• Sistema de aviso en caso de adormecimiento (Drowsy Driving Warning System). Determinan el grado de atención del conductor y le avisan en caso de que esté durmiendo.

• Control de velocidad variable (Adaptive Cruise Control) Se adapta la velocidad a la del vehículo que hay enfrente manteniendo la distancia de seguridad accionando el acelerador y el freno.

• Sistema anti colisión. (Anti Collision Assist) parecido al anterior pero ahora los obstáculos son coches parados, objetos en la vía, etc. Se avisa al usuario de la presencia de obstáculos o coches detenidos o a velocidades muy bajas.

• Parar y marchar (Stop & Go). Mantendrían el control del vehículo a bajas velocidades, por ejemplo en colas para entrar en las autopistas, en semáforos o en peajes.

• Sistema de ángulos muertos (Overtaking Warning) Los sensores cubren el ángulo muerto del vehículo avisando de la presencia de otros coches que estén realizando un adelantamiento.

• Alejamiento del lateral (Lane Departure Warning). El sistema detecta de forma automática la posición respecto a la línea lateral avisando al conductor si la va a sobrepasar de forma inadvertida. [11]

2.4 Sistemas inteligentes de detección de señales.

La detección de señales de tráfico ha desarrollado recientemente un interés en aumento por los grupos de investigación debido al desarrollo de la tecnología y sistemas de visión por computador. Esto se debe a que se pueden desarrollar diferentes aplicaciones como las siguientes:

- Mantenimiento de las señales en autopistas. Se sustituiría al operador personal que supervisa grabaciones de video para comprobar el estado de las señales.
- Inventario de señales en ciudades. Mismo motivo que el anterior pero en este caso al tratarse de ciudades existen muchos obstáculos y objetos con colores similares o incluso iguales que no facilitan la labor.
- Sistemas de ayuda a la conducción. Gracias a la lectura de la señal se puede ajustar la velocidad del vehículo, en el caso de las señales de velocidad, y también la trayectoria del mismo para el caso de otras señales.

Ante todo esto nos encontramos con tres dificultades que son las siguientes:

- Debido a que se trata de un entorno real al aire libre, dependemos de las condiciones atmosféricas para los niveles de iluminación ya que son cambiantes y no es posible controlarlos.
- En el entorno real no solo existen señales, puede darse el caso que algún objeto tape parte de la señal como por ejemplo una rama de un árbol, o la presencia de sobras que hagan que la señal no se aprecie del todo bien.
- Hay un rango de formas a determinar en las señales buscadas demasiado grande. [12]

Hoy en día existen algunos sistemas de detección incorporados ya en vehículos como por ejemplo el Ford S-MAX.

Para activar esta función los conductores tan solo tendrán que ajustar unos controles disponibles en el volante. El sistema utilizará la cámara ubicada en el parabrisas para detectar cualquier señal de tráfico que avise del límite de velocidad de la zona, por lo tanto se trata de un detector únicamente de señales de velocidad. Una vez detecte la señal automáticamente limitará el coche a dicha velocidad.

El sistema tiene una pequeña limitación ya que funcionará entre límites de 30 km/h y 200 km/h pero es suficiente ya que por encima y por debajo de esas velocidades no es normal la circulación. Lo que el sistema hace es que en vez de frenar drásticamente el coche si detecta una señal que reduce el límite de velocidad lo que hará es ir reduciendo velocidad de forma suave. En caso de que queramos que el limitador de velocidad no se active basta con pisar firmemente el pedal del acelerador para que se desactive la función. [13]

Figura 6: Panel de velocidad de un Ford S-MAX.

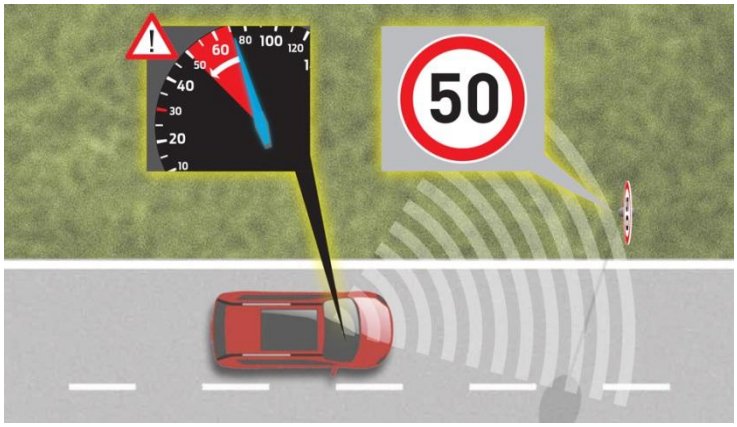

Figura 7: Representación de una reducción por detección de señal.

2.5 El coche autónomo de Google.

El coche autónomo de Google es un proyecto que consiste en desarrollar toda la tecnología necesaria para que un coche sea capaz de circular sin conductor de manera autónoma. Actualmente el líder del proyecto es el ingeniero alemán de Google Sebastian Thrun, director del Stanford Artificial Intelligence Laboratory y coinventor de Google Street View. El equipo de Thrun en Stanford creó el vehículo robótico Stanley, que fue el ganador del DARPA Grand Challenge en 2005, otorgado por el Departamento de Defensa de los Estados Unidos y dotado con un premio de 2 millones de dólares. Éste equipo estaba formado por 15 ingenieros de Google, entre los que se encontraban Chris Urmson, Mike Montemerlo, and Anthony Levandowski, quienes habían trabajado en el DARPA Grand and Urban Challenges.

Gracias a la detección de otros vehículos, señales de tráfico, peatones y otros muchos patrones de la carretera, este coche es capaz de conducir de forma autónoma tanto por ciudad como por carretera.

La circulación autónoma es posible ya que el estado de Nevada aprobó el 29 de junio de 2011 una ley que permite la operación de coches sin conductor, en la que Google habría presionado para el establecimiento de este tipo de leyes. La ley entró en vigor el 1 de marzo de 2012 y la primera licencia otorgada para un vehículo autónomo fue para un Toyota Prius modificado con la tecnología experimental driverless de Google.

A continuación se muestra una ilustración del coche autónomo de Google actual, con una breve explicación de algunos de los sensores que utiliza. [14]

Figura 8: Coche autónomo de Google.

El coche funciona como si de un conductor físico se tratara, constantemente se pregunta situaciones las cuales requieren de una respuesta:

<u>¿Dónde estoy?</u>

El coche es capaz de procesar la información procedente de los mapas y sensores para determinar en qué punto del mundo está situado, con tanta precisión que sabe en qué calle está en todo momento.

<u>¿Qué hay alrededor?</u>

Los sensores detectan todo aquello que está en su alrededor. El software los clasifica dependiendo de su tamaño, forma y patrones de movimiento. Por ejemplo es capaz de diferenciar entre un ciclista y un peatón.

<u>¿Qué va a suceder?</u>

El software que el coche lleva integrado es capaz de predecir que van a hacer los objetos que nos rodean, por ejemplo si un peatón va a cruzar la calle, o si un ciclista va a seguir con su marcha.

¿Qué hacer al respecto?

El software elige entonces una velocidad y una trayectoria de seguridad, y se aleja de la trayectoria del ciclista, o reduce la velocidad por si el peatón decide cruzar la calle.

Para poder entender cómo funciona el coche autónomo es necesario tener en cuenta que componentes y elementos tiene. Éstos se dividen en componentes físicos y en el software equipado.

Google empezó añadiendo componentes a coches normales como el Lexus SUV, para posteriormente quitar elementos innecesarios como por ejemplo los pedales y el volante, diseñando entonces un prototipo que se adapte a los elementos y al software como el que se ve a continuación:

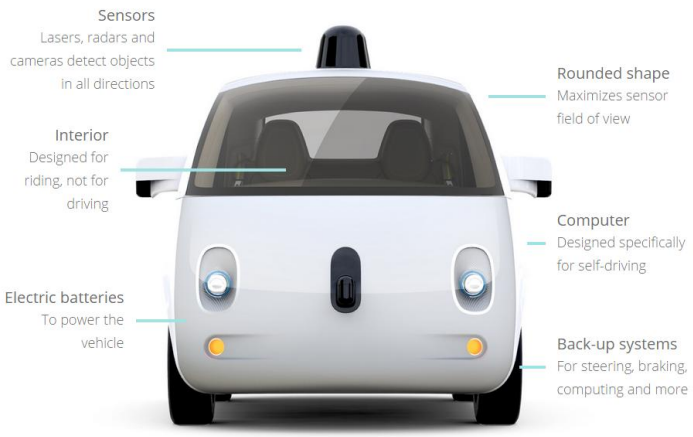

Figura 9: Coche autónomo de Google 2.

Merece la pena echar un vistazo a los proyectos del coche autónomo de Google desde el comienzo:

- Google comenzó a trabajar en su proyecto en 2009, probando la tecnología de conducción autónoma con un Toyota Prius en California:

Figura 10: Toyota Prius de Google.

- En 2012 se comenzó a probar en el Lexus RX450h llegando en este momento a más de 300 mil millas de pruebas de conducción libre.

Figura 11: Lexus RX450h de Google.

- Entonces llegó el momento de introducirlo en las calles de las ciudades, con un entorno mucho más complejo que las carreteras.

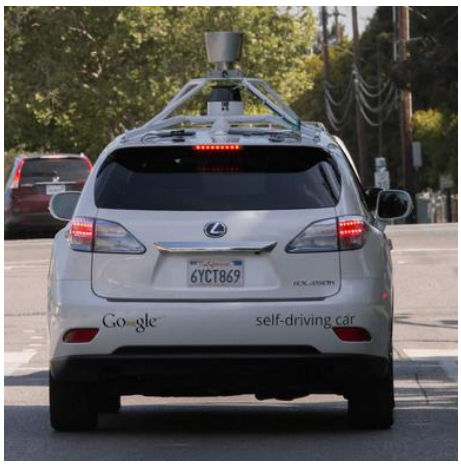

Figura 12: Lexus RX450h de Google, en ciudad.

- En 2014 se dio a conocer un prototipo de coche autónomo diseñado desde el principio para ser un coche totalmente de auto conducción.

Figura 13: Prototipo 1 de Google.

- Unos meses después, tras muchas pruebas se llevó a cabo la construcción del primer prototipo real en Diciembre del año 2014.

Figura 14: Prototipo 2 de Google.

Hoy en día el coche autónomo de Google ha recorrido más de 1 millón de millas y actualmente está en circulación por las calles de Mountain View, California and Austin, Texas.

La flota de pruebas va desde los Lexus modificados hasta prototipos de vehículos diseñados especialmente para la auto conducción. De momento en todos ellos hay conductores de seguridad con el fin de aprender acerca de las reacciones y posibles situaciones que se den con las personas reales y otros coches. [15]

2.6 Seguridad vial.

Para hablar de la seguridad vial es necesario distinguir en este proyecto entre dos apartados. En primer lugar los tipos de seguridad existentes, entre los que están la seguridad activa y la pasiva; y por otro lado los tipos de señales que están presentes hoy en día en nuestras carreteras:

2.6.1 Tipos de seguridad.

- **Seguridad activa:**

Es toda aquella seguridad que ayuda al conductor a evitar un accidente, interviniendo de forma permanente durante la circulación. Hay muchos sistemas de seguridad activa, entre ellos destacan los siguientes:

- Sistema de retrovisores: gracias a este sistema el conductor es capaz de ver lo que ocurre en la zona posterior del vehículo. Consta de espejos, supresión de puntos ciegos, radares y otros elementos con un gran avance tecnológico como la comunicación inalámbrica del vehículo y sistema de visión nocturna.
- Sistema de suspensión: aporta comodidad al vehículo, disminuye la transmisión de irregularidades en el terreno al habitáculo del vehículo. Además favorece el agarre e incrementa la estabilidad. Consta del mecanismo de amortiguadores.
- Sistema de frenado: este sistema es el encargado de disminuir la velocidad del vehículo durante su marcha. Esta disminución se produce mediante el rozamiento del freno, que puede ser de de dos tipos: de tambor o de disco.

  El frenado debe asegurar una rápida detección de las ruedas, pero sin llegar a bloquearse. Ante este fenómeno hay que tener en cuenta factores como el estado de la vía y el estado de los mecanismos del vehículo.

  Existen distintos tipos que optimizan la frenada y aumentan la controlabilidad del vehículo: ABS (Antilock Brake System: sistema anti-bloqueo de frenos) con EBV (reparto electrónico de frenada). ESP (control de estabilidad), con EDL (control de tracción).

- Sistema de dirección: el conductor del vehículo es el encargado de orientar la dirección del mismo mediante el uso del volante. El sistema de dirección orienta las ruedas con precisión y suavidad. En caso de ser una dirección asistida el conductor tiene que hacer mucho menos esfuerzo sobre el volante gracias a un sistema hidráulico.

Además existen sistemas de dirección servo-asistida que ayudan a reducir el esfuerzo al girar y mantener una dirección correcta cuando se circula a altas velocidades.

Estos sistemas pretenden asegurar un perfecto control del vehículo, aunque la circulación se produzca en condiciones adversas.

- Sistema de iluminación: Sirven como comunicación entre las personas y el resto de vehículos. Se trata de un elemento fundamental en la seguridad activa.

- **Seguridad pasiva:**

Se trata de la seguridad que se encarga de en caso de accidente, minimizar las consecuencias negativas que se pueden producir cuando este es inevitable. Destacan los siguientes sistemas:

- Cierre automático de la inyección de combustible para impedir incendios que puede afectar a cualquier persona.
- Depósito de combustible y elementos auxiliares diseñados para evitar el derrame de combustible en caso de colisión.
- Aviso automático a centro de emergencias después de un accidente.
- Puertas diseñadas para una fácil apertura después del accidente.
- Hebillas del cinturón de seguridad de fácil apertura.
- Llevar herramientas de seguridad en caso de emergencia.
- Pedalera colapsable: Minimiza los daños en las extremidades inferiores del conductor en caso de colisión frontal.
- Columna de dirección articulada colapsable: Esta columna cuenta con zonas de absorción de deformaciones que se localizan en la parte inferior del auto.
- Volante con absorción de energía: Donde la corona del volante y los radios son amplios y redondeados, cubiertos por un material deformable que no produce astillas.
- Parabrisas y cristales laterales: El compuesto utilizado en la fabricación del cristal parabrisas está preparado para que, en caso de accidente, no salten astillas que puedan dañar a los pasajeros del vehículo. En cambio, las ventanillas laterales son más débiles y pueden romperse más fácilmente, serían las salidas de emergencia en caso de volcado si las puertas quedasen bloqueadas. [16]

2.6.2 Señales y tipos de señales.

Conceptualmente se trata de todas aquellas señales además de las órdenes de los agentes de circulación y las señales circunstanciales que cambian el régimen normal de la vía y las señales fijas de balizamiento, semáforos, señales verticales de circulación y marcas viales, destinadas a los usuarios de la vía y que tienen una misión que es la de advertir e informar a

éstos u ordenar o reglamentar su comportamiento con la necesaria antelación de determinadas circunstancias de la vía o circulación.

La señalización constituye una parte esencial de todo sistema de circulación. Las señales sirven para transmitir al conductor ordenes e información acerca de normas de comportamiento; advertencias sobre peligros que pueden aparecer en la vía en la que se circula; y por último informaciones de todo tipo de interés para el conductor, con el propósito de facilitar su conducción.

Gracias a las mismas el usuario de las vías públicas conoce, en un momento determinado, las condiciones de uso de la vía por donde circula, y en consecuencia, puede afectar el recorrido o la maniobra adecuada para evitar retrasos o accidentes.

Así pues, la existencia de la señalización, se fundamenta en obtener la máxima seguridad y eficacia en el uso de las vías públicas.

Las señales son un sistema de comunicación en el que el lenguaje hablado es sustituido por el lenguaje de las formas, los colores, los símbolos o signos diversos. Mediante ellas se emiten mensajes a unos receptores que son los conductores y usuarios de las vías, los cuales deben responder con un comportamiento adecuado.

La principal misión de las señales es la de advertir, reglamentar o informar a los usuarios de la vía, con la necesaria antelación, de determinadas circunstancias de la misma o de la circulación.

Por tanto la señalización podemos decir que persigue tres objetivos:

- Aumentar la seguridad de la circulación.
- Aumentar la eficacia de la circulación.
- Aumentar la comodidad de la circulación.

De acuerdo con estos fines las señales cumplen las siguientes funciones:

- Advierten los posibles peligros.
- Ordenan la circulación, de acuerdo con las circunstancias del lugar.
- Recuerdan algunas prescripciones del Reglamento General de la Circulación.
- Proporciona al usuario de la vía una información conveniente. [17]

3. Software y Hardware.

3.1 Software.

3.1.1 Versión Android utilizada.

La versión utilizada e incorporada en el Smartphone es la versión Android 4.4 popularmente conocida con el nombre "Kit-Kat". Se trata de la API 19 cuya fecha de lanzamiento fue el 13 de Octubre del año 2013.

Históricamente Android ha tenido nombres clave para todas sus versiones desde abril de 2009, y sus nombres siguen un orden alfabético: Apple Pie, Banana Bread, Cupcake, Donut, Éclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich, Jelly Bean, KitKat (el nombre de una chocolatina de Nestlé) y Lollipop 5.1

A continuación se observa una gráfica de la distribución global de las distintas versiones del sistema operativo Android a lo largo del tiempo desde Diciembre del año 2009 hasta Enero del año 2015.

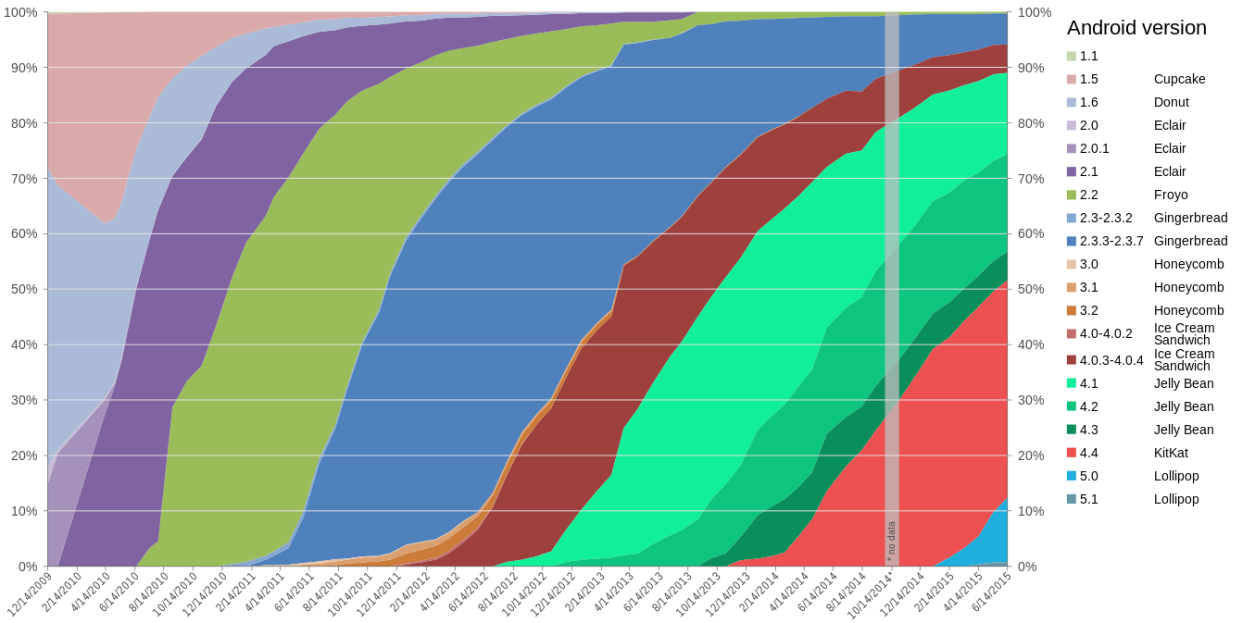

Figura 15: Distribuciones globales de las versiones de Android.

No obstante la importancia de las versiones viene en el momento de saber desde que versión está OpenCV asegurado en un correcto funcionamiento, que en este caso se trata de la versión Android 2.2 conocida como "Froyo" con API 10 lanzada en Mayo del año 2010. [18]

3.1.2 Versión OpenCV utilizada.

Para poder utilizar OpenCV es necesario que nuestro dispositivo móvil tenga una versión de Android 2.2 o superior. Para que las bibliotecas de OpenCV ejecuten correctamente se requiere que el Smartphone tenga cámara trasera y si es posible una frontal también, pero ésta última ya depende del uso de la aplicación que se quiera desarrollar. Posteriormente se describirá la cámara que el dispositivo Android utilizado tiene integrado, en el apartado siguiente "Hardware".

En caso de no tener otra opción se puede configurar para que el emulador interprete la webcam del ordenador como si fuera la cámara del Smartphone.

Respecto a la versión descargada y utilizada de OpenCV para la elaboración de este proyecto es la **2.4.9** descargada de la página oficial de OpenCV "*http://opencv.org/downloads.html*". Para poder trabajar con ésta librería se importa desde Eclipse el archivo que se encuentra en la carpeta SDK del archivo previamente mencionado.

El paquete de OpenCV contiene ejemplos de proyectos ya implementados como por ejemplo "*face-detection*", "*color blob detection*", "*15 puzzle*", "*image manipulations*"… que ejecutándolos correctamente permite ver diferentes aplicaciones que la biblioteca nos permite realizar. [19]

Figura 16: Android 4.4 KitKat.

3.1.3 Eclipse + Android SDK

Eclipse es un programa informático compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" que están basadas en navegadores. Esta plataforma, ha sido usada para desarrollar entornos de desarrollo integrados (IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el

compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse).

Eclipse es también una comunidad de usuarios, extendiendo constantemente las áreas de aplicación cubiertas. Un ejemplo es el recientemente creado Eclipse Modeling Project, cubriendo casi todas las áreas de Model Driven Engineering.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Hoy en día es desarrollado por la Fundación Eclipse, una organización que es independiente y que no tiene ningún ánimo de lucro. Además esta fundación fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

Hay que destacar que por un lado tenemos el software Eclipse y por otro tenemos el Android SDK. En primer lugar descargamos e instalamos ambos software independientes para luego desde Eclipse vincular el Android SDK mediante un tercer software llamado ADT (Android Developent Tools). Una vez terminado el proceso ya tenemos vinculado el SDK con la plataforma Eclipse.

Es entonces cuando podemos empezar a tratar el código y desarrollar la aplicación. La plataforma Eclipse cuenta junto con el SDK con un emulador para que en caso de no disponer de un terminal donde probar los proyectos, poder probarlos en el mismo ordenador, pero desde OpenCV no está recomendado el uso de este emulador ya que hay algunas funciones que no ejecutan de manera correcta. [20]

3.1.4 App BaseLSI.

Para desarrollar el proyecto de la detección de señales hay que tener en cuenta que se trabaja para el departamento de Sistemas Inteligentes y en éste caso se ha facilitado una aplicación base a partir de la cual diferentes alumnos han desarrollado sus proyectos añadiéndolos a ésta misma. Podría decirse que se trata de una aplicación de la cual salen subaplicaciones con diferentes funciones cada una. Entre otras podemos encontrar aplicaciones para detectar peatones, coches y distancias entre vehículos.

Todas ellas en su conjunto pueden llevar en un futuro cuando se trate de una herramienta completa a una aplicación que contiene todo lo necesario y relacionado con la conducción de un vehículo autónomo.

En el menú principal de la aplicación se encuentra la siguiente captura:

Figura 17: Captura LSI menú principal.

En donde aparecen cinco botones:

- Botón superior LSI: al pulsarlo direcciona a la URL de LSI (laboratorio de sistemas inteligentes).
- Aplicaciones LSI: En este botón están las aplicaciones desarrolladas por alumnos en diferentes proyectos con el departamento de sistemas inteligentes.
- Ejemplos: Esta sección tiene distintos ejemplos propios de la biblioteca importada OpenCV.
- ¿Quiénes somos?: Aquí se encuentra la información relacionada con el proyecto y el departamento de Sistemas Inteligentes.
- Universidad Carlos III de Madrid: Conduce a la web de la Uc3m.

Una vez pulsado el botón de **aplicaciones LSI** aparece un submenú con dos opciones como se ve a continuación en la siguiente captura:

Figura 18: Captura LSI aplicaciones.

- Ejecutar aplicación: pulsando este botón aparece otro submenú en donde se encuentran la lista con las aplicaciones ya realizadas.
- Volver: para volver al menú principal.

Pulsando **Ejecutar aplicación** aparece la lista con las aplicaciones ya realizadas, probadas e implementadas es la siguiente:

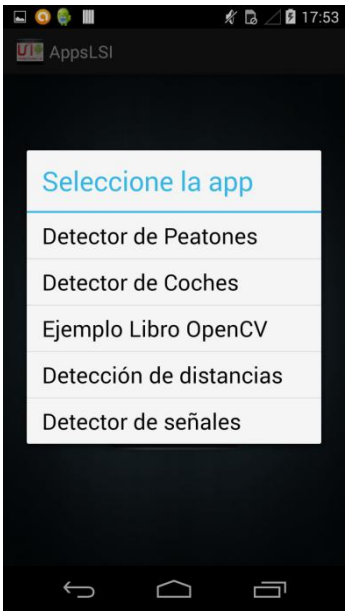

Figura 19: Captura LSI selección app.

- Detector de peatones: Se trata de una aplicación creada con el fin de detectar peatones en tiempo real. Ésta es capaz de distinguir peatones entre distintos tipos de objetos y obstáculos.
- Detector de coches: Al igual que el detector de peatones tiene la misma función pero para la detección de coches.
- Ejemplo Libro OpenCV: Se trata de una aplicación que permite capturar imágenes, editarlas y almacenarlas en la memoria interna del dispositivo, es un ejemplo ya creado por OpenCV.
- Detección de distancias: Permite calcular y tener uso de los datos de las distancias de objetos cercanos al dispositivo móvil, o en este caso a la cámara del mismo.
- Detector de señales: es la aplicación implementada en este proyecto de fin de grado.

Volviendo de nuevo al menú principal se pulsa en el segundo botón **Ejemplos:**

Figura 20: Captura LSI ejemplos OpenCV.

En este submenú aparecen tres botones que de nuevo ejecutaran tres nuevas aplicaciones, que en este caso son ya ejemplos implementados por la biblioteca OpenCV pero que han sido incluidas en la aplicación base para poder manipular las imágenes de entrada y poder conseguir fines determinados con la misma. Con estos ejemplos se pueden hacer desde zoom a la imagen de entrada hasta resolver un puzle que OpenCV ha incluido en sus ejemplos:

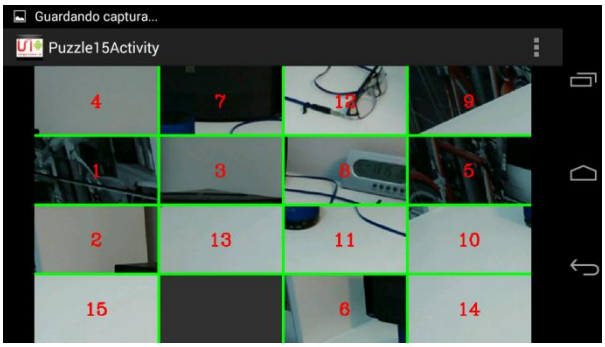

Figura 21: Captura LSI ejemplo puzle OpenCV.

Por último accediendo al botón **¿Quiénes somos?** Se muestra la información relativa al desarrollo de la aplicación y al departamento de sistemas inteligentes.

Figura 22: Captura LSI quienes somos.

Todo esto hace el total de la aplicación base LSI desarrollada previamente por el departamento. [21]

3.2 Hardware.

3.2.1 Modelo de Smartphone utilizado.

Para el desarrollo del proyecto se ha precisado de un Smartphone, el utilizado ha sido el MOTOROLA MOTO G de la primera generación, ya que posteriormente han aparecido modelos renovados de este mismo.

Fue adquirido en el año 2013 por lo tanto es un terminal relativamente nuevo que incorpora un sistema operativo Android 4.4 (KitKat) previamente mencionado que permite ejecutar perfectamente el proyecto diseñado.

Como características importantes de éste Smartphone hay que destacar la capacidad de almacenamiento que a pesar de no tener opción de insertar una memoria externa mini-SD posee una capacidad de 6 GB de almacenamiento interno. Esto es clave ya que es en este espacio en donde se almacenan las imágenes de señales de tráfico a modo de base de datos para poder reconocerlas. Estas imágenes ocupan muy poco pero a mayor capacidad, mayor número de imágenes podremos introducir.

Otra característica importante es la conectividad al ordenador portátil, ya que para probar y ejecutar la aplicación desde eclipse es necesario cargarla al Smartphone mediante un cable USB.

Por último un elemento importante en este Smartphone es la cámara gracias a la cual obtenemos imágenes en vivo que posteriormente trataremos y modificaremos para llegar al resultado deseado.

Figura 23: Smartphone utilizado, Motorola Moto G.

3.2.2 Características de la cámara utilizada.

Este dispositivo tiene dos cámaras, una trasera y otra delantera.

La cámara trasera tiene una resolución de 5 MP que a pesar de no ser una resolución muy alta comparado con los Smartphone más actuales, permite trabajar perfectamente la imagen de entrada.

Por otro lado la cámara delantera tiene una resolución de 1,3 MP que puede ser utilizada también en el proyecto si fuera necesario y se podría trabajar con las dos cámaras. [22]

4. Diseño y desarrollo del proyecto.

4.1 Planteamiento del proyecto.

El proyecto se plantea con el fin de realizar una aplicación, para el sistema operativo Android, que sea capaz de detectar señales de tráfico.

Podría haberse desarrollado para otra plataforma como por ejemplo el sistema operativo iOS pero se ha hecho en Android ya que está mucho más desarrollado y trabajado tanto el software Eclipse como el OpenCV en ésta.

Basándonos en la idea de la conducción autónoma, el proyecto tiene el fin de traer de una manera fácil y sencilla al usuario la opción de realizar una detección de señales de tráfico, al igual que lo hacen los coches autónomos, a pesar de que éstos van integrados con multitud de cámaras y sensores que un Smartphone no dispone. Por así decirlo, facilita el acceso a ésta experiencia al usuario de calle, ya que solo tendría que descargarse la aplicación del Google Play Store.

Una correcta lectura de una señal de tráfico podría evitar muchos accidentes ya que un coche autónomo podría cumplirlas de una manera estricta mientras que los conductores físicos en muchas ocasiones no las cumplen. Un ejemplo claro es el de las señales de velocidad. Este es uno de los problemas actuales en las carreteras que más accidentes provoca. El exceso de velocidad sigue siendo la causa de uno de cada cuatro fallecidos en accidente de tráfico.

Cumplir con los límites legales de velocidad podría evitar una cuarta parte de los muertos en accidente de tráfico. En 2013 en nuestro país, 366 personas fallecieron y 1.518 resultaron heridos graves en accidentes en los que la velocidad fue uno de los factores concurrentes. 233 de las muertes y 958 de los heridos graves lo fueron en accidentes ocurridos en vías convencionales.

	Fallecidos 30D	H. graves 30D
Autopista + Autovía	46	163
Vías convencionales	233	958
Vías urbanas	87	397
Total	366	1518

Figura 24: Tabla de fallecidos y heridos graves en accidentes con factor velocidad, año 2013.

Según el Informe sobre la situación mundial de la seguridad vial 2013 de la Organización Mundial de la Salud, la limitación legal de la velocidad y su observancia pueden reducir de forma significativa los accidentes y las lesiones causadas por el tráfico.

Pese a que la mayoría de los conductores circulan a la velocidad establecida, todavía hay más de un millón de conductores que el año pasado en los controles practicados por la Agrupación de Tráfico de la Guardia Civil excedían los límites de velocidad.

Esta cifra va en la misma línea del estudio que la DGT realizó sobre velocidad libre en las carreteras españolas, es decir, velocidad a la que circulan cuando hay muy poco tráfico, nula presencia policial o de vigilancia, buen tiempo y tramos rectos, con ausencia de entradas y salidas y con poco o nulo desnivel. En dicho estudio se destaca que:

· En carreteras convencionales el 39% de los vehículos superaban la velocidad límite establecida y el 13% superaban el límite en más de 20 kilómetros por hora, lo que significa el doble de vehículos registrados con este exceso que en las vías desdobladas, en donde el porcentaje de vehículos que rebasaban el límite en 20km/h fue del 6,5%. En estas vías convencionales es donde se producen el 70% de los accidentes con víctimas

· En autovías, el 10% de los conductores de vehículos superaban en 10 kilómetros la velocidad límite establecida

Otro estudio a nivel europeo, SARTRE 3, financiado por la UE sobre comportamientos y actitudes sociales, estima que el 25% de todos ciudadanos de la UE admiten superar los límites de velocidad en las autopistas y autovías y el 13% en las carreteras convencionales.

Respecto a los peatones, un informe de la OMS establece que a partir de 80km/h es prácticamente imposible que un peatón se salve en un atropello. A una velocidad de 30km/h, el riesgo de muerte del peatón se reduce al 10%. [23]

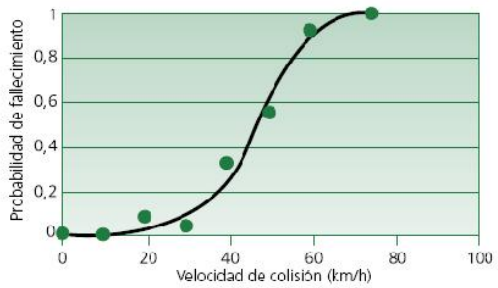

Figura 25: Gráfica de riesgo de fallecimiento de un peatón en función de la velocidad de colisión de un vehículo.

Ante este último problema de atropello a peatones, otra de las aplicaciones que ayudan a la detección de los mismos evitaría muchos de los atropellos producidos por parte de vehículos a

peatones, ya que detendrían instantáneamente la marcha del vehículo, acortando así el tiempo de reacción del conductor físico.

4.2 Desarrollo del proyecto.

Una vez se tiene claro el objetivo y las causas que dan lugar al proyecto, que en este caso es la creación de una aplicación, se puede entonces proceder al desarrollo del mismo.

Para explicar el desarrollo de la app se ha dividido el desarrollo en varios apartados.

4.2.1 Imagen de entrada.

En primer lugar y lo más importante, para poder detectar señales es necesario tener un lugar en donde buscarlas. Este "lugar" es una situación real que es capturada por nuestra cámara del Smartphone.

La imagen de entrada podría ser también una imagen por defecto introducida por código en Eclipse, en la cual se encontraría la señal presente en dicha imagen. Pero como lo que nos interesa es detectar señales en tiempo real, se realiza con la captura en vivo con la cámara.

La app busca entonces continuamente señales en cada fotograma o frame que la cámara captura.

Figura 26: Imagen de entrada con señal de 120 km/h.

La estructura del código utilizada se basa en la clase básica **Class CameraBridgeViewBase** que es la encargada de interactuar entre la cámara y la biblioteca OpenCV. Su función principal es controlar cuando la cámara debe ser activada o desactivada, el proceso de los fotogramas de entrada, y llama a los procesos externos para poder modificar la imagen de entrada y posteriormente sacarla por pantalla. Se debe implementar con el CvCameraViewListener. [24]

La clase CvCameraViewListener tiene los siguientes métodos y descriptores:

- OnCameraFrame (Mat inputFrame): Este método es el corazón de la clase. Se invoca cuando se necesita que el fotograma de entrada sea resuelto. El valor de retorno es el fotograma que queramos que se muestre en la pantalla.
- onCameraViewStarted (int width, int height): Se invoca éste método cuando se inicializa la cámara. Una vez se ha invocado los fotogramas son entregados al cliente a través de la llamada a onCameraFrame ().
- onCameraViewStopped (): se invoca éste método cuando la vista de la cámara ha sido detenida por algún motivo. Después de llamar a éste método no se mandarán fotogramas a onCameraFrame ().[25]

4.2.2 Procesamiento de la imagen.

Gracias a la biblioteca importada OpenCV se puede acceder a métodos que nos permiten procesar y manipular la imagen de entrada según se necesiten. Todas estas operaciones dependerán del fin deseado ya que cada una funciona de un modo, y hay que tener siempre un conocimiento teórico previo para saber cómo funcionan.

Se puede decir que cada operación que se realiza a los fotogramas es un paso más para llegar a resolver el problema que se desea trabajar.

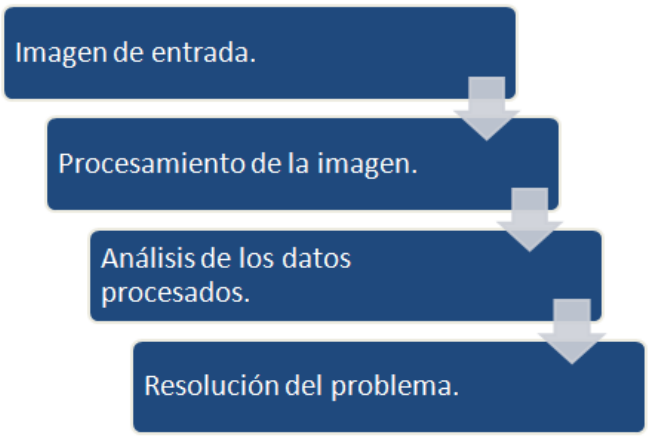

Figura 27: Diagrama funcionamiento del procesamiento de datos.

Dentro del apartado del procesamiento a continuación se describen los algoritmos utilizados para llegar al análisis de la información de entrada en forma de imagen.

4.2.2.1 Algoritmos utilizados.

Una vez se tienen los fotogramas capturados por la cámara en vivo, llega el momento de manipular éstos fotogramas. En primer lugar es mucho más fácil trabajar con una imagen en espacio de color HSV, teniendo en cuenta que la imagen viene por defecto en RGB hay que convertirlo.

HSV proviene del inglés Hue, Saturation, Value y corresponde a la Matiz, Saturación y Brillo respectivamente. Es un modelo de color definido por sus componentes.

Siempre que se tenga una imagen es normal que se elija el espacio de color adecuado para ésta, para ello resulta muy útil usar la ruleta de color HSV. En ella el matiz se representa por una región circular; una región triangular separada, puede ser usada para representar la saturación y el valor del color. Normalmente, el eje horizontal del triángulo denota la saturación, mientras que el eje vertical corresponde al valor del color.

De este modo, un color puede ser elegido al tomar primero el matiz de una región circular, y después seleccionar la saturación y el valor del color deseados de la región triangular. [26]

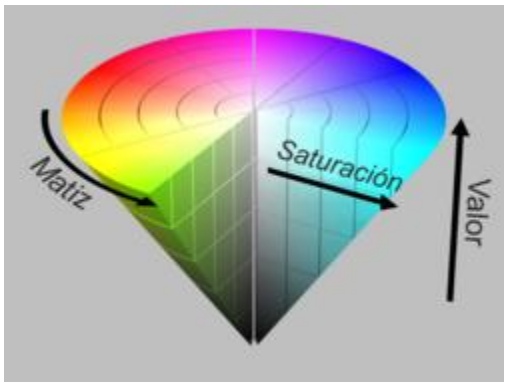

Figura 28: Cono del modelo HSV.

Para realizar la conversión de RGB a HSV OpenCV ofrece una clase llamada *Imgproc* en la cual hay cantidad de métodos e identificadores. En este caso utilizaremos el método siguiente:

- **cvtColor(Mat src, Mat dst, int code)**: en donde Mat src es la matriz de imagen de entrada, que en este caso es inputframe; Mat dst es la Mat de imagen destino, en donde se almacena la imagen una vez convertida; y por último y más importante, el

code es el código de identificación de la conversión que se va a realizar, en este caso al ser src una Mat RGB y al querer una Mar dts en espacio HSV el código será *RGB2HSV*.

Figura 29: Imagen de entrada en RGBA.

En esta imagen se observa como la captura de entrada se realiza en un espacio de colores RGB, que a continuación se convierte con el algoritmo ya explicado a HSV y se obtiene el siguiente espacio de color:

Figura 30: Imagen de entrada en HSV.

Una vez se tiene la imagen en HSV es necesario determinar qué color se va a buscar en ella, ya que una buena forma de detectar objetos en las imágenes es buscar colores en la misma. Teniendo en cuenta que en las señales redondas de prohibición existe un borde rojo en el contorno que ayuda a detectar las de este tipo, al mismo tiempo que las azules por ejemplo son toda la señal azul excepto la figura central.

Por eso para buscar colores se ha utilizado el siguiente método:

- **inRange(Mat src, Scalar lowerb, Scalar upperb, Mat dst)**: se establece un umbral de un color para determinar que pixeles se quieren como 1 y cuales como 0. Esto es así ya que la salida de éste método es una Mat binaria que da como 1 el pixel de color que se comprende entre el umbral determinado, y 0 todos aquellos que no lo estén. De éste modo ya se tiene una Mat con fotogramas en los que solo se trabaja con los píxeles del color buscado.
  Respecto a los parámetros que tiene éste método, trabajamos con una Mat de entrada que en este caso es la previamente convertida a color HSV, dos vectores escalares previamente definidos, uno para el valor alto y otro para el valor bajo (valores expresados para el espacio de color HSV); y por último la Mat destino que es en donde se almacenará los fotogramas después de éste método.

Se observa que en el ejemplo anterior de la imagen de entrada aplicando un filtro para el color azul dentro de un rango determinado por código se obtiene la siguiente imagen:

Figura 31: Imagen de entrada con filtro de color azul.

Y aplicándolo a una señal de tráfico con componentes azules se obtiene los siguientes resultados:

Figura 32: Señal azul de entrada en RGB.

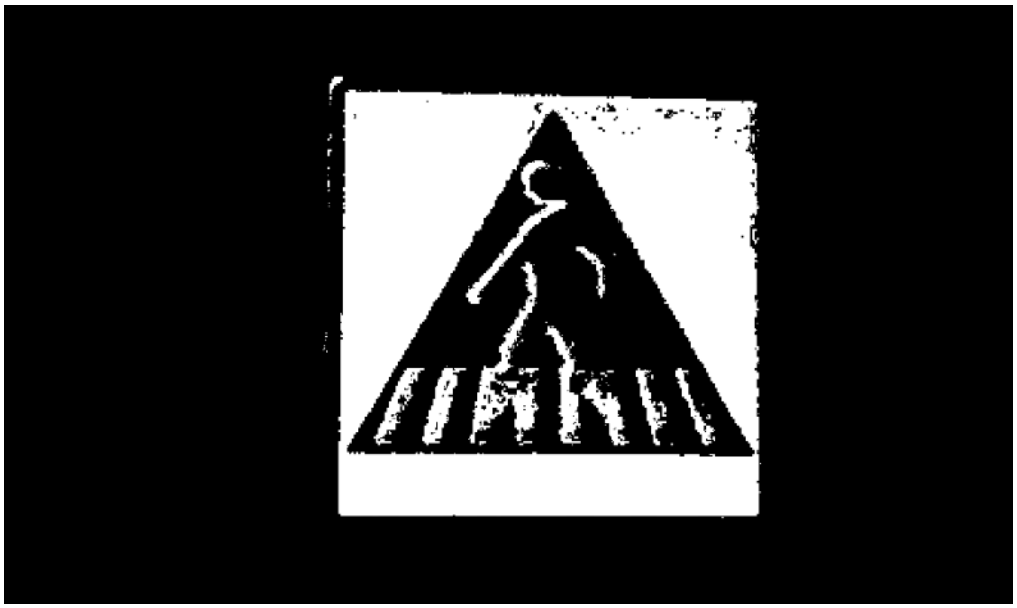

Figura 33: Señal azul de entrada con filtro en rango de color azul.

Se aprecia que toma algunos pixeles de la figura central que son negros como azules, este es uno de los inconvenientes ya que la iluminación y la situación real de la cámara afectan a estos rangos.

Este mismo caso se puede ver y aplicar para las señales con componente roja:

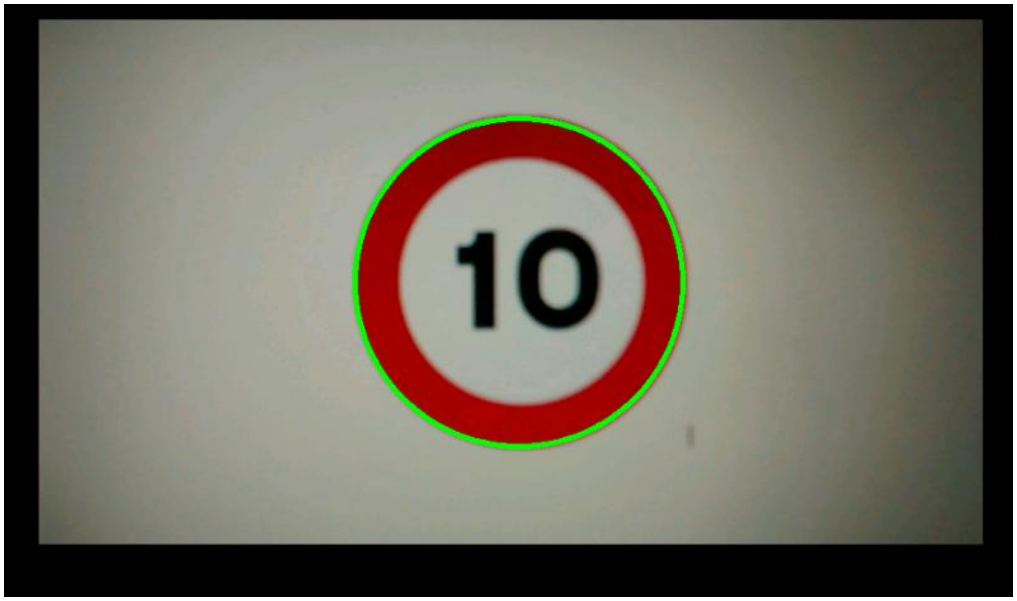

Figura 34: Señal velocidad de entrada en RGB.

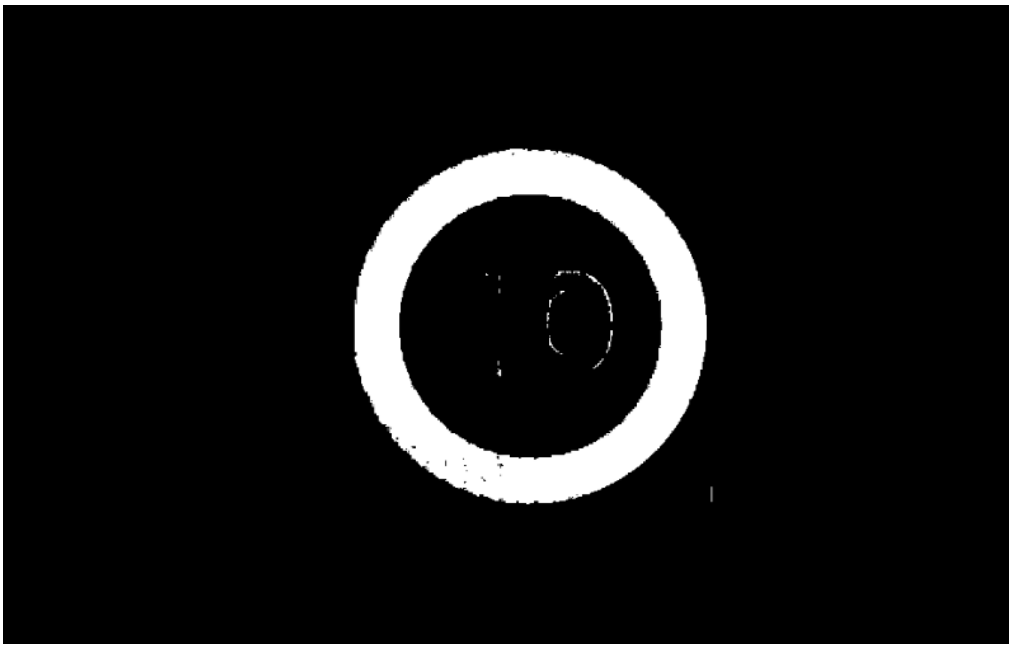

Figura 35: Señal roja de entrada con filtro en rango de color rojo.

El siguiente método a implementar es una dilatación para facilitar la posterior búsqueda de contornos de los círculos.

- **dilate (Mat src, Mat dst, Mat kernel)**: esta función dilata la imagen usando un elemento estructurante específico. Los parámetros de entrada son la Mat src que es la

misma que la de la salida de la función inRange previa; la Mat dst que es la de salida de ésta función, es decir, la destino; y por último Mat kernel que es el elemento estructurante del cual depende la dilatación.

Se tiene entonces un fotograma con los colores deseados dilatados, para encontrar los contornos es necesario hacer una operación más. Para ello hacemos una resta de fotogramas, en la cual se restan las imágenes quedándose solo los bordes buscados. Para ello se utiliza la función absdiff perteneciente a la clase Core.

- **Absdiff (Mat src1,Mat src2,Mat dst)**: con éste método se realiza la resta entre dos fotogramas. El primer fotograma o Mat corresponde a la Mat de entrada src1, el segundo a la Mat de entrada src2 y la resta se almacena en la Mat de salida dst, con la que se trabaja posteriormente.

Una vez se ha realizado las operaciones de dilatación y resta se puede ver el resultado a continuación:

Figura 36: Resultado del fotograma después de dilatación y absdiff para color azul.

Figura 37: Resultado del fotograma después de dilatación y absdiff en señal para color rojo.

Esto ayuda a que la detección de círculos dentro de la imagen sea más limpia y favorezca a la supresión de la detección de falsos círculos.

4.2.2.2 Identificación señales redondas.

Una vez se tiene la resta en la cual existen solo los pixeles blancos que corresponden al borde del color buscado, llega el momento de buscar círculos en los fotogramas para poder encontrar las señales de tráfico redondas, ya sean las de prohibición que corresponden a aquellas que tienen el marco rojo, o las de obligación que son las azules.

Figura 38: Ejemplo señal 10 km/h.

Figura 39: Ejemplo señal obligación.

Un paso previo a la detección de círculos es el de de aplicar un filtro gaussiano para eliminar el ruido existente en el fotograma, y de este modo evitar la detección de falsos círculos. Se trata de una función que pertenece a la clase Imgproc.

- **GaussianBlur (Mat src, Mat dst, Size ksize, double sigmaX, double sigmaY)**: filtra la imagen con un filtro gaussiano, lo realiza haciendo una convolución de la imagen de entrada, que en este caso es Mat src, guardándolo en la de salida Mat dst. El parámetro ksize es el tamaño del núcleo gaussiano, mientras que sigmaX y sigmaY son las desviaciones estándar en las direcciones x e y respectivamente.

Una vez se ha evitado el ruido para la detección de falsos círculos, para encontrar los círculos deseados OpenCV cuenta con un método que pertenece a la clase Imgproc llamado Houghcircles.

- **HoughCircles(Mat image,Mat circles,int method,double dp,double minDist, double param1, double param2, int minRadius,int maxRadius)**: esta función busca círculos en una imagen de entrada con escala de grises. En este caso ésta imagen de entrada es la de salida del filtro gaussiano y corresponde al parámetro Mat image; otros parámetros importantes a describir son el parámetro Mat circles que es un vector en el que se almacenan los círculos encontrados; minDist que determina la mínima distancia que ha de existir entre los centros de dos cirulos encontrados; minRadius y maxRadius que determinan el radio mínimo y máximo que han de tener los círculos buscados.

Una vez el programa ha sido capaz de encontrar los círculos, para mejorar la experiencia con el usuario, se ha decidido dibujar éstos círculos. Para ello se ha recurrido a la clase Core y dentro de la misma a la función circle.

- **Circle(Mat img, Point center, int radius, Scalar color, int thickness)**: ésta función dibuja un circulo en la imagen deseada, que en este caso es la Mat img que corresponde a aquella que queramos que aparezca por pantalla en el Smartphone, por lo tanto es la imagen de entrada, inputFrame. Para que la función dibuje el círculo es necesario determinar el centro del mismo, el radio, el color y el grosor del círculo

dibujado. El centro y el radio se obtienen de los parámetros de los círculos obtenidos y guardados en la Mat circles del paso previo, mientras que el color y el grosor se pueden definir en la declaración de la función.

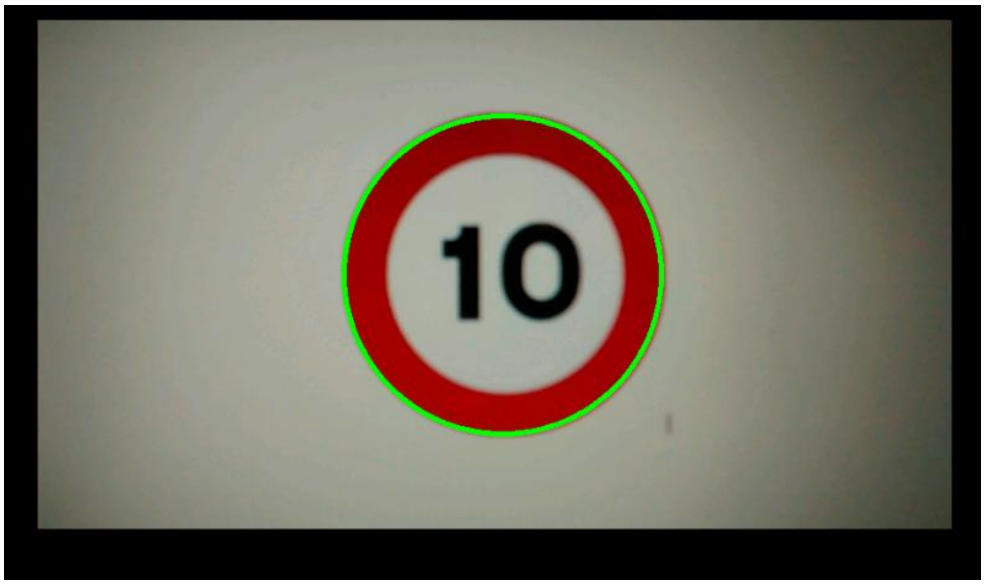

Figura 40: Detección y dibujo del círculo en señal de velocidad.

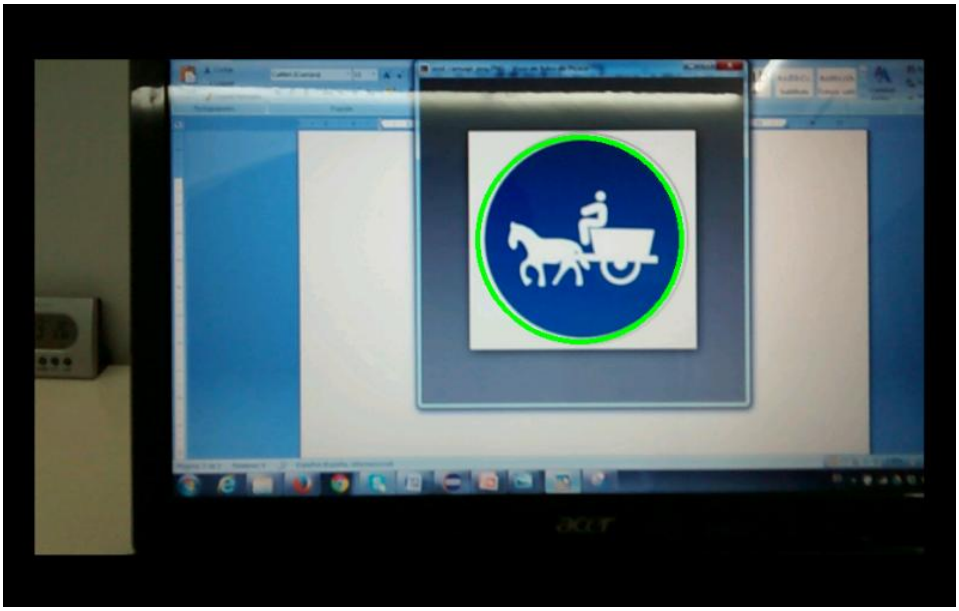

Figura 41: Detección y dibujo del círculo en señal azul de carruaje.

Una vez se tiene el círculo buscado en los fotogramas de entrada, se procede al recorte de ésta región de interés para hacer la identificación posterior por características comunes de ambas imágenes. Para ello se crea una función llamada *recorta_círculos* en la cual realiza el siguiente proceso:

- Se tiene el círculo buscado, en caso de detectar más de un círculo la app se queda con el de mayor radio y que cumpla las condiciones de radio máximo y mínimo.
- Cumplidas estas condiciones se determina las dimensiones de la zona a recortar, que en éste caso se hace gracias a una nueva clase Rect que utiliza una plantilla para rectángulos en 2 dimensiones. A este constructor hay que pasarle varios parámetros que son la altura, anchura, y dos coordenadas x e y que corresponden a la posición de la esquina superior izquierda del mismo, quedando así Rect(int x, int y, int widht, int height).
- Para determinar los parámetros del paso previo se accede al vector de cirulos encontrados y se definen gracias al radio de los mismos.
- Una vez definidas las dimensiones del rectángulo a recortar hay que comprobar que no son mayores que la imagen de entrada (inputFrame), y en caso de serlo se reajustan las dimensiones.
- Por último y una vez cumplidos todos los pasos previos, se crea una nueva Mat de fotogramas en la que se tiene como imagen la señal existente en la imagen de entrada (inputFrame, imagen en vivo real) y dimensiones del rectángulo a recortar.

Lo siguiente que la aplicación realiza es la comparación e identificación de características entre el recorte y las imágenes almacenadas en el Smartphone. Para que la app pueda identificar éstos recortes deberá tener una carpeta en la memoria con bastantes imágenes de señales y poder hacer la comparación de características y puntos clave. [27]

Éste proceso se describe detalladamente en el punto 4.2.4 identificación de señal.

4.2.2.3 Identificación de otras señales.

Lo que la aplicación efectúa nada más iniciarse es la búsqueda de las señales circulares previamente descrita. En caso de no encontrar ninguna de este tipo, busca cualquier otra señal, por ejemplo triangulares o cuadradas.

Figura 42: Ejemplo señal triangular.

Figura 43: Ejemplo señal cuadrada.

Para la identificación de éste tipo de señales se accede directamente a la búsqueda bruta de señales en toda la imagen de entrada. Se diferencia del método anterior en que en el anterior la imagen es tratada y procesada hasta que se encuentra un círculo. Para la búsqueda de estas otras señales la imagen no se trata ni modifica sus pixeles.

El método utilizado es al igual que en las circulares ya recortadas, la identificación y comparación de las características y puntos clave de la imagen de entrada con las imágenes almacenadas en la memoria del Smartphone.

4.2.3 Diseño de las funciones.

A lo largo del desarrollo del desarrollo del código ha sido necesario crear varias funciones. En primer lugar unas funciones en las que se pudiera umbralizar el color buscado, siempre dentro del espacio de color HSV, y se han creado dos funciones una para el color rojo y otra para el azul. Destacar aquí que también se creó en un primer lugar una última umbralización para el color amarillo pero que se descartó posteriormente ya que el círculo rojo de la señal que rodea al color amarillo se detecta de forma suficiente con el umbral de la función roja, y en el caso de las señales con fondo amarillo cuadradas, se detectan correctamente con la detección bruta.

Una segunda función creada ha sido la previamente descrita para buscar círculos, llamada ***busca_circulos*** en la que el código se ejecuta para la búsqueda y clasificación de los mismos dentro de la imagen de entrada.

Una vez se tienen los círculos se creó la función ***recorta_circulos*** que lo que hace es recortar los círculos encontrados para después hacer la identificación de señales.

Además de estas funciones creadas para resolver el problema de las señales, se han creado también otras para inicializar la biblioteca OpenCV llamada BaseLoaderCallback() cuya función es la de cargar el OpenCV Manager.

La función onCreate() permite cargar la actividad con el valor de la superficie de trabajo. Otra función creada es la función onPause() que mantiene la cámara con la superficie creada para cuando la actividad sea pausada, mientras que onResume() se ejecuta cuando cargue la actividad ya que se debe realizar una inicialización asíncrona de OpenCV. La función onDestroy() sirve para desactivar la vista una vez que la actividad se pause o se destruya, y por último la función reiniciarCamara() que actualiza los parámetros de la cama cuando sea reiniciada.

Todas estas funciones sumadas a las implementadas previamente propias de la clase CvCameraViewListener hacen que la aplicación funcione correctamente. Para llamar a las funciones es necesario hacerlo desde onCameraFrame() ya que es el corazón de la aplicación.

4.2.3.1 Buscar círculos.

Esta función ha sido descrita previamente en el apartado de identificación de señales redondas, no obstante se va a explicar ciertos detalles de cómo funciona.

Una vez la imagen ha sido tratada y tenemos una imagen en escala de grises en la cual solo tenemos como pixeles blancos los bordes de los objetos con color rojo/azul, entramos en el proceso de buscar círculos con la función houghcircle, esta función lo que hace es almacenar los círculos encontrados en una matriz. Una vez todos los círculos encontrados han sido almacenados, se accede a dicha matriz y teniendo en cuenta la posición del centro y el radio del mismo, se procede a dibujar el contorno del círculo en señal de que el cirulo ha sido encontrado.

4.2.3.2 Recortar círculos encontrados.

Esta función también ha sido explicada previamente. Se detalla a continuación algunos aspectos.

Una vez se ha identificado y guardado el círculo en la matriz de círculos, éste se recorta y entonces se analiza solo lo que hay dentro de este nuevo recorte. Para llevar a cabo este recorte de la región de interés es necesario que el código haya almacenado el radio y el punto del centro del círculo. Al mismo tiempo se han diseñado un buce para que el recorte de éste círculo sea siempre de dimensiones adecuadas.

Una vez se ha realizado el recorte se extrae la componente roja de la imagen para entonces pasar a la identificación de la señal con solo los elementos que hay dentro del círculo.

4.2.4 Identificación de la señal.

La identificación de la señal se realiza de dos modos. En primer lugar el programa busca señales circulares en la imagen de entrada. Por otro lado si no encuentra ningún círculo, se realiza la identificación bruta de señales para poder identificar cualquier otro tipo de señal. Se ha optado por este proceso para poder trabajar con los fotogramas de la imagen de entrada y poder elaborar de este modo el trabajo realizado para identificar las señales circulares.

Una vez se ha dibujado el contorno del círculo encontrado y recortado la región de interés buscada que resulta ser la de la señal circular, se procede a la identificación de ésta señal.

Para ello se ha creado una clase llamada ImageMatcher que se explica en el siguiente apartado. Lo que ésta clase es capaz de realizar es gracias a los puntos clave de la imagen, accede a la memoria interna del dispositivo y saca coincidencias entre la imagen capturada y todas las imágenes almacenadas de las señales existentes en la memoria interna del dispositivo móvil (Smartphone).

Aquí se aprecia cómo funciona ésta identificación:

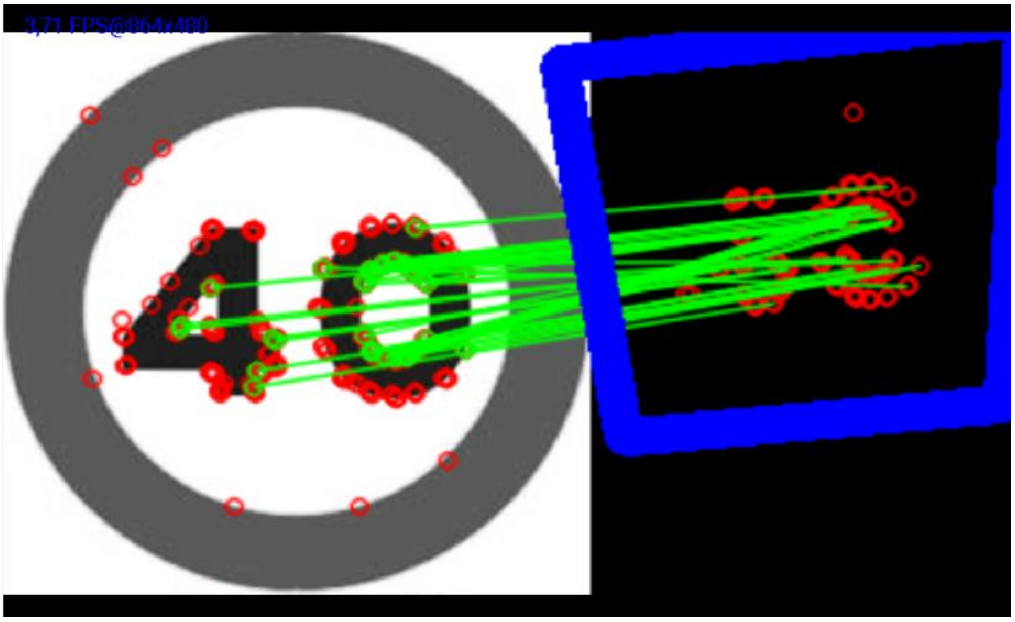

Figura 44: Visión de identificación de una señal de 40 km/h.

Una vez se realiza esta coincidencia de puntos, si se da por satisfactoria la identificación de la señal se ha diseñado a la derecha de la imagen de captura de la pantalla un apartado en el cuál se muestra la señal en caso de ser identificada:

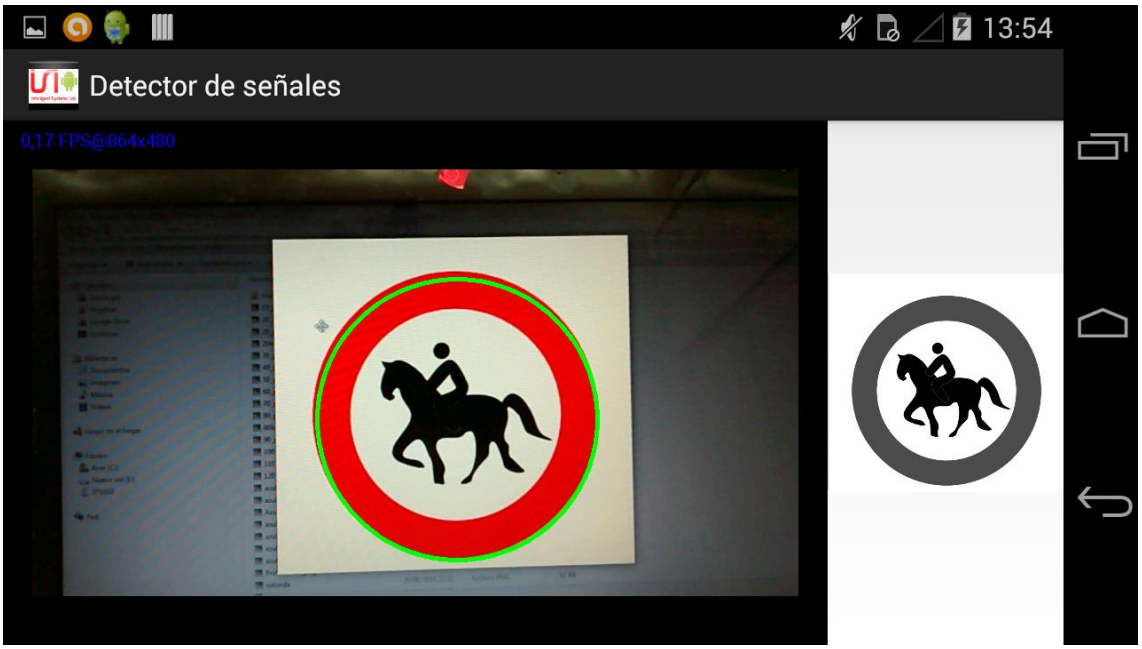

Figura 45: Detección de señal con respectiva visión de la misma.

4.2.4.1 Clase ImageMatcher.

La clase ImageMatcher ha sido creada e incluida en la aplicación con el fin de poder llevar a cabo la identificación de las señales. Es la clase encargada de ver, analizar, y decir si la señal ha sido identificada correctamente o no.

En primer lugar antes de entrar en detalle con la clase es necesario explicar otras clases base del tipo abstractas que han sido utilizadas para poder llevar a cabo la identificación de imágenes:

Feature Detector:

El detector de características se utiliza para encontrar puntos clave de un objeto en una imagen de entrada.

Dentro de la detección de características y puntos clave existen distintos métodos que han sido empleados a lo largo del código de la clase ImageMatcher. El método ***create*** permite crear un detector y llamarlo como se desee. También ha sido utilizado el método ***detect*** que detecta los puntos clave en una imagen o en un conjunto de imágenes.

Por otro lado existen distintos tipos de detectores (FAST, STAR, BRIEF, SURF, ORB…) y el utilizado en éste caso ha sido el ORB.

La clase ORB se implementa para la detección de puntos clave y para la extracción de descriptores. El algoritmo utiliza la clase FAST en pirámides para detectar puntos clave estables, selecciona los más fuertes con FAST, encuentra su orientación utilizando momentos de primer orden y calcula los descriptores usando BRIEF. [28]

Descriptor Extractor:

Se trata de una clase base abstracta utilizada para calcular descriptores de puntos clave de una imagen. Dentro de esta clase hay métodos que han sido utilizados como el método ***create*** gracias al cual se puede crear un descriptor por su nombre. Al igual que FeatureDetector la implementación de éste método soporta distintos tipos de extractores (SIFT, SURF, BRIEF, ORB…). También se ha utilizado el método ***compute*** que calcula los descriptores para un conjunto de puntos clave dentro de una imagen o conjunto de imágenes. [29]

Una vez se han detectado y extraído las características y los puntos clave de las imágenes, se llega al punto en el cual se hace el emparejamiento de los mismos, es decir, la comparación de los de una imagen con la otra. Para ello se hace uso de la clase DescriptorMatcher:

Descriptor Matcher:

Es la clase base para emparejar o hacer coincidir los descriptores de los puntos clave, cuenta con dos grupos de métodos de emparejamiento: para hacer coincidir los descriptores de una imagen con otra imagen, o por otro lado con un conjunto de imágenes. Para definir la clase y nombrarla se ha empleado el método ***create*** que crea el emparejador de descriptores de un determinado tipo con los parámetros por defecto, es decir, usando un constructor por defecto. En este caso se ha empleado el tipo BRUTEFORCE que es un tipo de emparejador de descriptores que determina emparejamientos de puntos clave similares. Además se ha utilizado el método ***match*** que encuentra la mejor combinación para cada descriptor de un conjunto de consultas sucesivas. [30]

Teniendo en cuenta estas tres clases gracias a las cuales se puede llevar a cabo la identificación de las señales de tráfico, es importante explicar cómo se accede a la carpeta de imágenes que se ha creado dentro de la memoria interna del Smartphone.

Para ello es necesario tener acceso a las variables del entorno y acceder al directorio de la carpeta con la función *getExternalStorageDirectory () +"nombre de la carpeta"*. Además se ha creado un filtro para que se procesen solo las imágenes, es decir, los archivos que estén en ésta carpeta y terminen en .jpg o en .png. Para ello se ha utilizado el interfaz *FilenameFilter* que se basa en los nombre de los archivos como ya se ha explicado.

De forma resumida lo que esta clase efectúa es lo siguiente:

En primer lugar se crea un detector de características y un extractor de descriptores para la imagen de entrada es decir la imagen capturada por la cámara del Smartphone. Éstos se analizan y se almacenan para posteriormente comparar con las imágenes almacenadas en la memoria del Smartphone, las cuales también han pasado por este proceso de detectar características, extraerlas, y emparejarlas con la imagen capturada.

Una vez se ha realizado el proceso se devuelve a la clase principal null en caso de no existir emparejamiento o una Mat con los puntos emparejados que corresponden a la imagen encontrada, gracias a los cuales desde la clase principal *SignalDetector* se establece y se representa la señal como identificada en la parte derecha de la pantalla del Smartphone.

Para hacer el emparejamiento se hace mediante la función Match. Se ha diseñado el código para que se pueda ver también las líneas de emparejamiento, para eso la función Match tiene dos variables que son la imagen de entrada y un Boolean que si se pone en true permite ver en la imagen derecha las líneas de emparejamiento entre ambas imágenes, es decir, la de captura en vivo de la señal recortada y la señal almacenada, y si se deja en false se ve solamente la señal identificada.

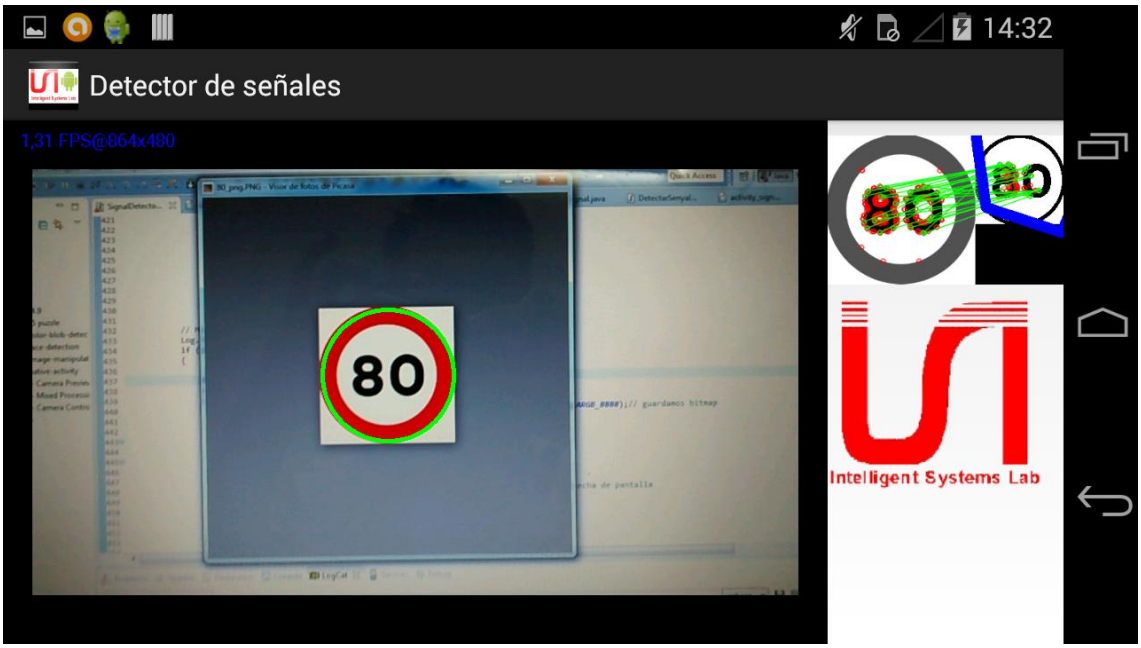

Figura 46: Visión de las trazas de emparejamiento en señal de velocidad.

4.2.4.2 Llamada desde clase principal.

Para poder acceder a la clase creada ImageMatcher es necesario hacerlo desde la clase principal, que en este caso ha sido llamada "***SignalDetector***".

Se debe invocar a la clase ImageMatcher desde la clase principal siempre que se quiera realizar una identificación de señales.

Para ello se ha de declarar previamente la clase que se llamará posteriormente con la función Match y si nos devuelve algo que no sea cero, este algo se trata de una Mat con una imagen. Esta imagen es necesario convertirla a mapa de bits que posteriormente se sitúa y representa en la parte derecha de la pantalla en su correspondiente símbolo de LSI.

4.2.5 Modos de ejecución.

Para elegir el modo de ejecución se ha creado una variable de tipo entero "mode" cuyo valor se realiza por asignación en las líneas de código, o en un menú de opciones desplegable creado en la parte superior de la aplicación. En función del valor de este entero se puede ejecutar la aplicación de dos modos:

- Cuando mode=1 o en el menú de la aplicación se selecciona DETECCION BRUTA se ejecuta el código de identificación de las señales de forma bruta, es decir, se busca señales en toda la imagen de entrada. Este algoritmo es rápido y encuentra todo tipo de señales (siempre y cuando se encuentren en la memoria del Smartphone). En este caso la señal se representa en la parte superior derecha de la pantalla, es decir en el logo superior de LSI, el icono inferior en este modo no se utiliza.
- Cuando mode!=1 o en el menú de la aplicación se selecciona DETECCION PROCESADA se ejecuta el algoritmo de detección de señales con previa manipulación de la imagen de entrada. Este método es algo más lento, pero detecta de forma correcta todas las señales al igual que el primero.

  Además dentro de éste modo se han creado dos opciones con una variable llamada p, solo modificable en código:

  - Si p=1 se va a buscar en la imagen solo las señales redondas, ya sean azules o rojas.
  - Si p=2 se va a buscar todo tipo de señales.

Por defecto la aplicación está en mode!=1 y p=2 para poder identificar todas las señales con el procesado previo de imagen en las circulares. Si se desea una mayor rapidez en la detección es necesario cambiar a mode=1 o DETECCION BRUTA para que la imagen no se procese y simplemente se haga la detección bruta de las mismas.

En resumen la diferencia entre ambos métodos consiste en que el primer método es más rápido pero no procesa la imagen, mientras que el segundo procesa la imagen la manipula y encuentra las señales también pero con una manipulación, análisis y extracción de datos de los fotogramas. Ambos métodos son totalmente aptos y válidos para buscar señales, que es el fin de ésta aplicación.

4.2.5.1 Identificación de señal bruta de imagen.

Se ejecuta cuando la variable mode es igual a 1 o cuando se ha seleccionado DETECCION BRUTA. Convierte la imagen de entrada inputFrame en una Mat a escala de grises y hace una búsqueda de señales recurriendo a la llamada de la clase imageMatcher con la Mat de escala de grises recién creada. Una vez se ejecuta todo el algoritmo hallado en la clase imageMatcher en caso de encontrar coincidencias con las imágenes que están en la memoria del Smartphone, se representa la señal correspondiente con dicha coincidencia a la derecha de la pantalla del Smartphone, y en caso de no encontrarlas no lo hace.

En este caso la identificación y la confirmación de que la imagen ha sido detectada se realizan en la parte superior derecha de la pantalla:

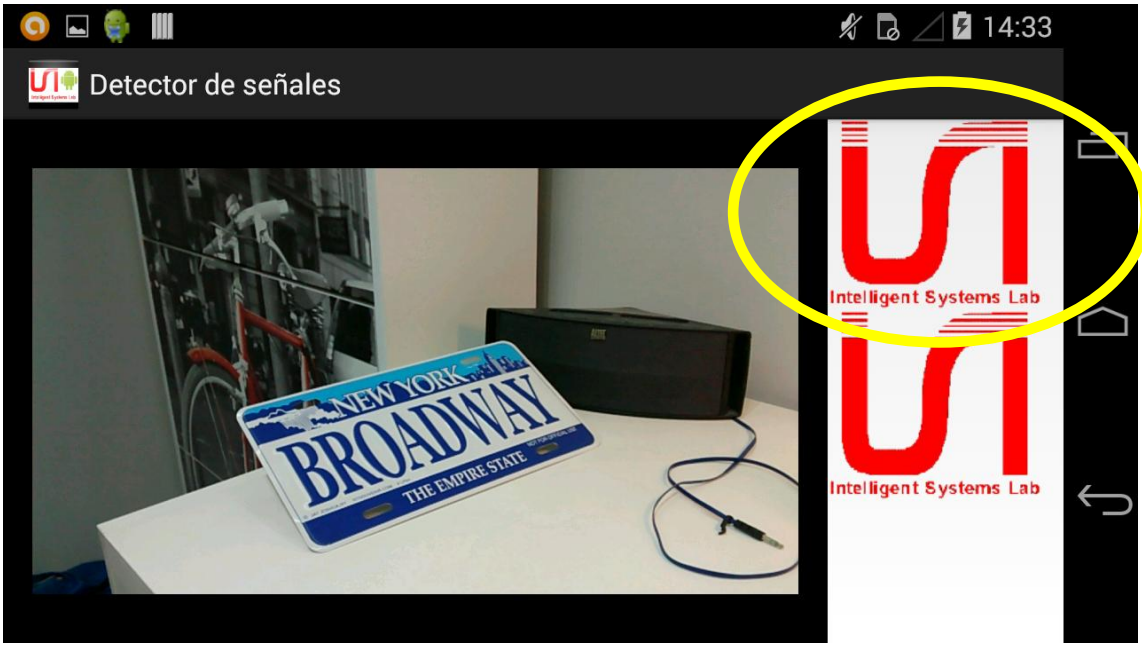

Figura 47: Visión general de app en referencia a identificación bruta.

A continuación se observa un ejemplo de este tipo de identificación, en el cual la aplicación ha buscado señales circulares, y al no encontrarlas ha realizado la identificación bruta en la que ha encontrado una señal triangular:

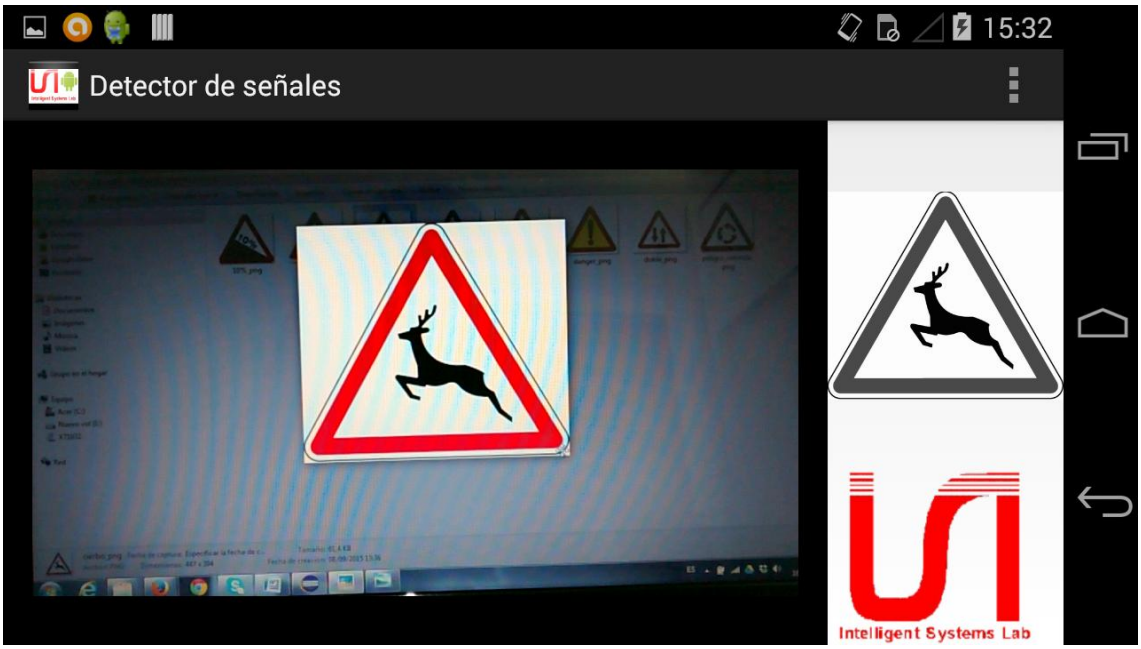

Figura 48: Detección de señal triangular por identificación bruta.

4.2.5.2 Identificación de señal con manipulación previa de imagen.

Se ejecuta cuando la variable mode es distinta de 1 o cuando se ha seleccionado DETECCION PROCESADA. Entonces la tarea que realiza la app es modificar y procesar la imagen de entrada mediante procesos descritos previamente como en el paso 4.2.2.2. Una vez se ha conseguido llegar al paso final de estas manipulaciones se accede a la clase imageMatcher del mismo modo que en mode=1 y se ejecuta de la misma manera, es decir, se ejecutan todas las líneas de código presentes en dicha clase y si éste encuentra e identifica la señal, la representa en la parte derecha de la pantalla del Smartphone. Esta representación en este lugar es solo para las señales circulares que han sido procesadas:

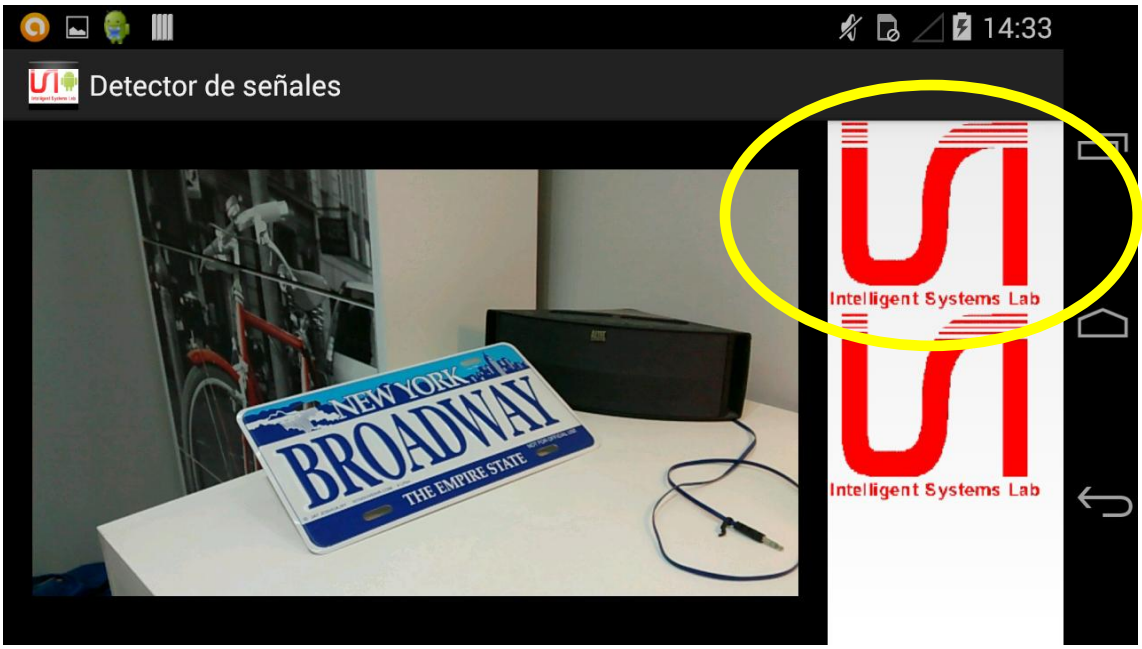

Figura 49: Visión general de app en referencia a señal circular.

A continuación se muestra la identificación de una señal de velocidad de 40 km/h:

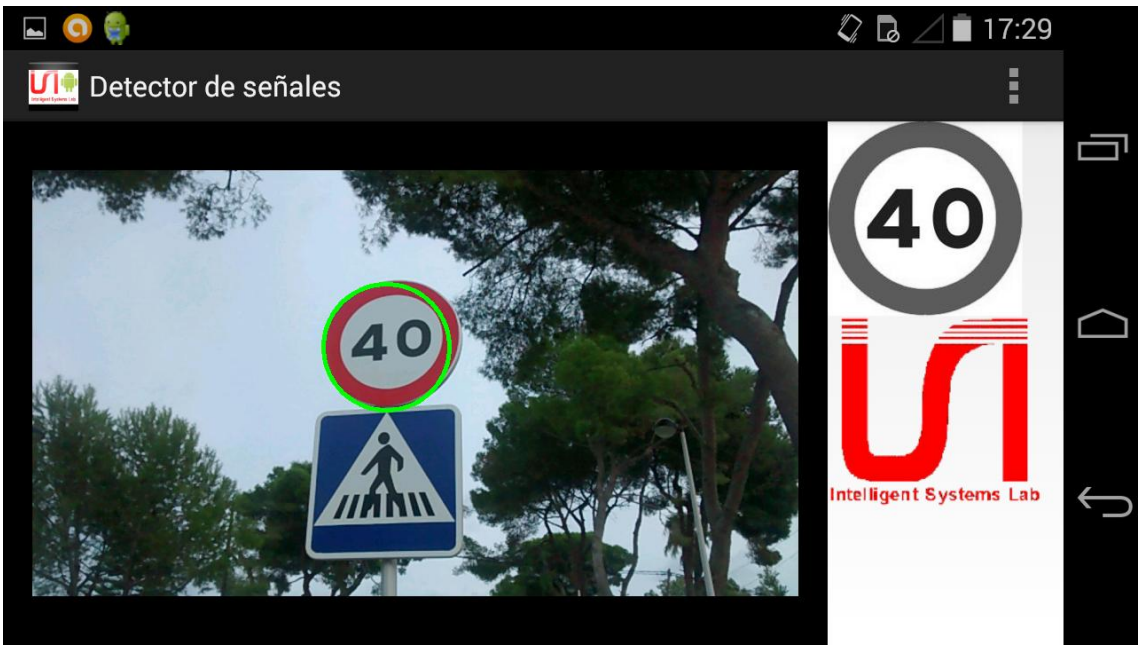

Figura 50: Detección de señal de velocidad con manipulación de fotograma.

Existe una limitación a éste modo, que consiste en que tan solo se ha conseguido manipular y procesar la imagen de entrada para encontrar señales redondas. Para hacer la identificación de las señales triangulares o cuadradas se recurre a la identificación bruta, es decir, se ejecuta una condición que dice que si no se encuentra ninguna señal redonda, acceda la aplicación entonces al código de detección bruta para detectar algún otro tipo de señal como las triangulares o cuadradas.

De este modo si la iluminación o alguna otra circunstancia que hace que no se detecte las señales redondas, se acceda también a la detección bruta asegurándonos una detección de las señales.

En este caso la representación de la señal encontrada se realiza en la parte inferior derecha, es decir el icono LSI de abajo:

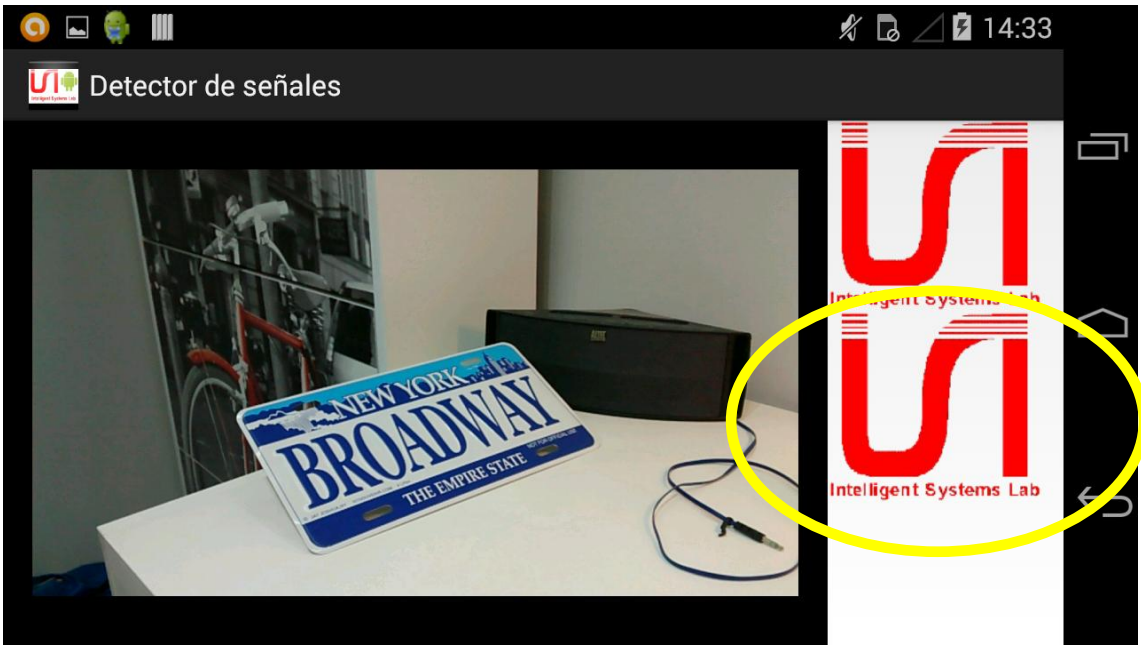

Figura 51: Visión general de app en referencia señal no circular en mode!=1.

A continuación se muestra un ejemplo de este caso:

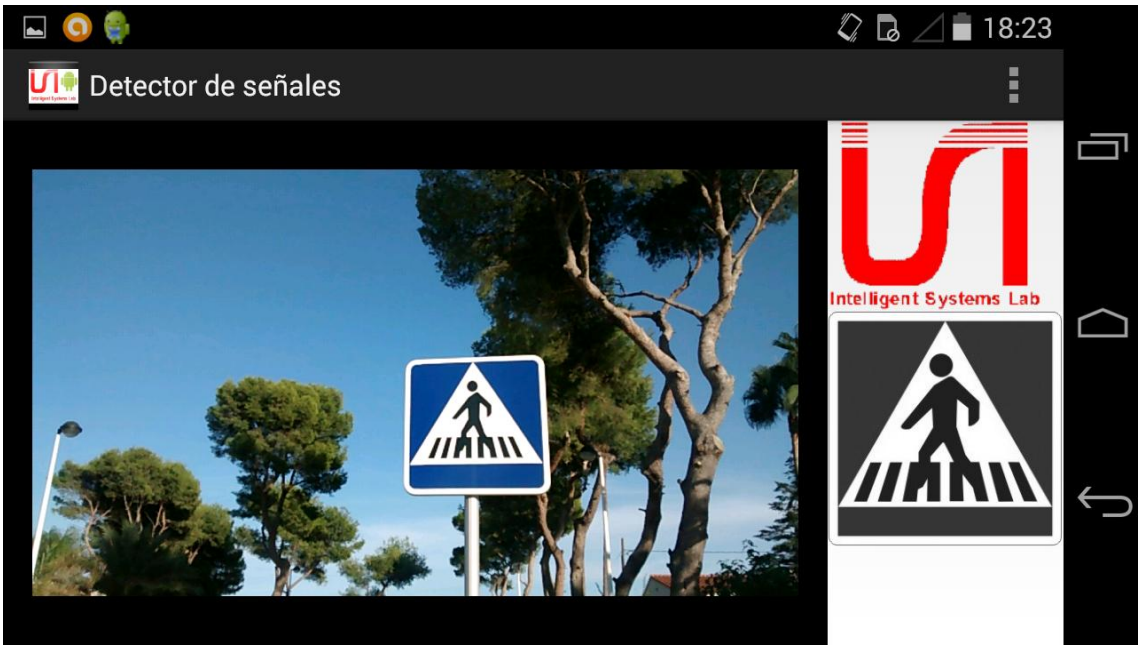

Figura 52: Detección de señal cuadrada en modo !=1.

Se observa que se trata de una señal cuadrada, por lo tanto la aplicación ha tratado de buscar círculos en la imagen y al no encontrarlos ha ejecutado la detección bruta identificando la señal y representándola en la parte inferior derecha previamente descrita.

Una vez explicados los modos de ejecución y el funcionamiento de la aplicación se presenta a continuación los flujogramas de ambos modos en los que puede funcionar la app.

Se presenta uno para cada modo es decir, para cuando se realiza la detección con procesado de imagen y cuando se realiza la detección bruta:

- **MODE=1 / DETECCION BRUTA:**

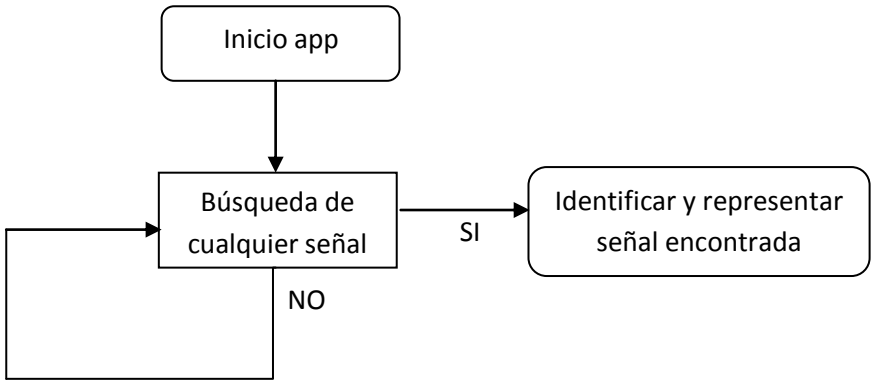

Figura 52: Flujograma del funcionamiento en mode=1.

- **MODE!=1 / DETECCION CON PROCESADA:**

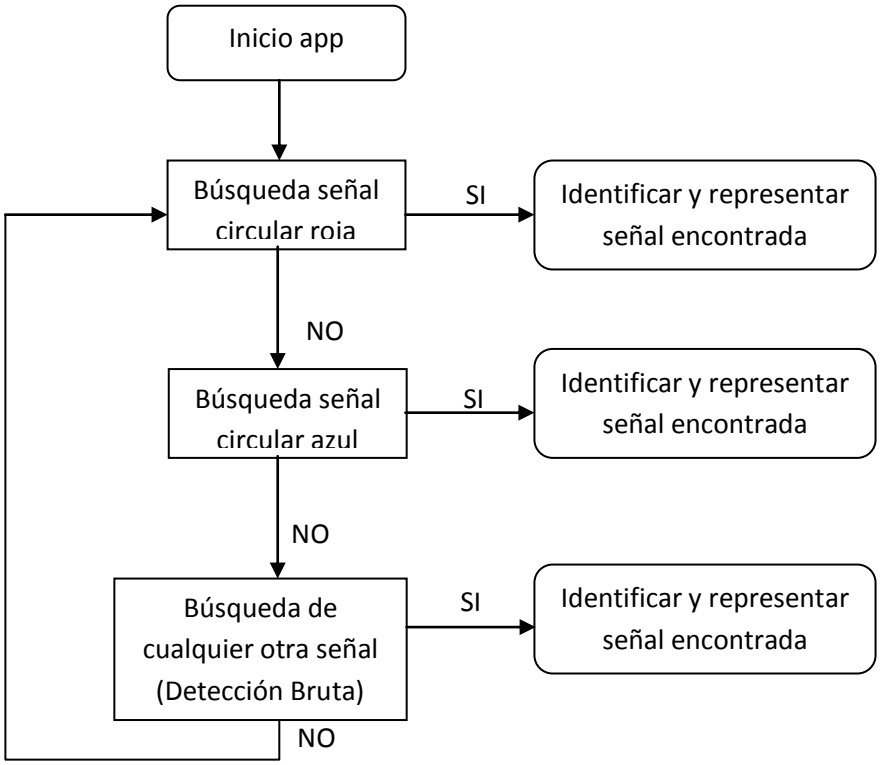

Figura 53: Flujograma del funcionamiento en mode!=1.

5. Pruebas y resultados.

5.1 Funcionamiento aplicación a nivel usuario.

La aplicación como ya se ha explicado está diseñada y pensada para que el usuario pueda entrar en la tienda de su Smartphone y descargarse la aplicación. Una vez descargada e instalada la aplicación puede ser usada por cualquier tipo de usuario que desee identificar señales.

El usuario no tiene acceso a la manipulación del código, por lo tanto solo puede identificar señales. Para poder modificar el modo de funcionamiento, como ya se ha explicado puede elegir en el desplegable creado entre la detección bruta y la detección con manipulación de fotograma. Este menú lo que hace es cambiar de valor la variable ***mode*** dentro del código.

El modo por defecto es el "DETECCION PROCESADA" que es el que manipula y procesa los fotogramas antes de realizar la identificación (mode!=1).

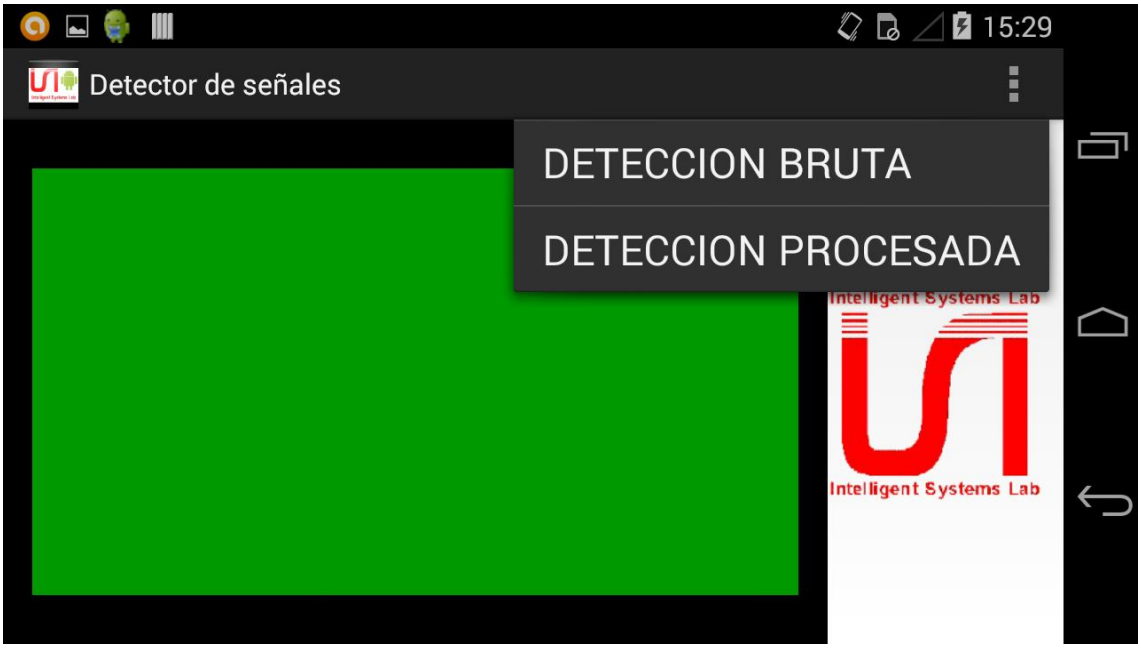

Figura 54: Modos de funcionamiento a nivel usuario.

El usuario debe colocar su dispositivo en el salpicadero del vehículo y abrirla y disfrutar de la misma mientras conduce y circula por la carretera.

Figura 55: Posición de Smartphone en salpicadero del coche.

5.2 Análisis de las pruebas.

A lo largo de la creación de la aplicación ha sido necesario ir probándola para ver y saber que se estaba llegando al resultado adecuado. En primer lugar la prueba de crear la aplicación desde cero se ha ido realizando desde el paso de crear con Eclipse la representación gráfica de lo que va a ser la aplicación, es decir el entorno visual en donde se ve las imágenes de la app. Posteriormente se ha creado el código que ha sido posible ir probándolo gracias a que la aplicación ya había sido creada. Una vez todo ha sido creado se ha procedido a realizar las pruebas para analizarlas y poder sacar conclusiones.

5.3 Análisis de los resultados.

Los resultados obtenidos después de las pruebas finales han sido bastante satisfactorios. En primer lugar tener en cuenta que con la detección bruta directa la aplicación es capaz de identificar cualquier señal, siempre y cuando esté dentro de la base de datos, es decir, la carpeta de la memoria interna del Smartphone.

Por otro lado las pruebas en el modo de procesado de la imagen para las señales circulares han sido satisfactorias también. La aplicación es capaz de detectar círculos tanto rojos como azules y una vez detectados y rodeados con una circunferencia de color verde, es capaz de identificarlos, en ocasiones no de forma satisfactoria pero ese apartado se explicará en el

siguiente punto. En este modo también se identifican el resto de señales ya que se recurre a la detección bruta en caso de no encontrar círculos en la imagen. Por lo tanto es un modo que funciona bien y soluciona el problema que llevó a la creación de esta aplicación, la detección de señales desde un Smartphone Android.

5.4 Limitaciones y problemas.

A la hora de realizar las pruebas de la aplicación han aparecido varias limitaciones y problemas que se describen en este apartado, todos los problemas descritos se refieren al modo de DETECCION PROCESADA, o mode!=1.

- En primer lugar en el modo de ejecución en el cual se procesa la imagen buscando círculos la imagen en el Smartphone es muy lenta. En ocasiones limpiando y refrescando el código desde Eclipse se consigue que funcione más fluido pero muy lento comparado con el otro modo.
- A la hora de identificar las señales en las que aparecen dígitos o números se presenta un problema. La clase ImageMatcher se queda con los puntos importantes y significativos que existen dentro de la señal. Por lo tanto en las señales de velocidad al tratarse de números en muchas ocasiones se queda con puntos significativos comunes en muchas de las señales, e identifica la señal con otra de otra velocidad. Aquí se aprecia éste problema en distintos casos que se explican:

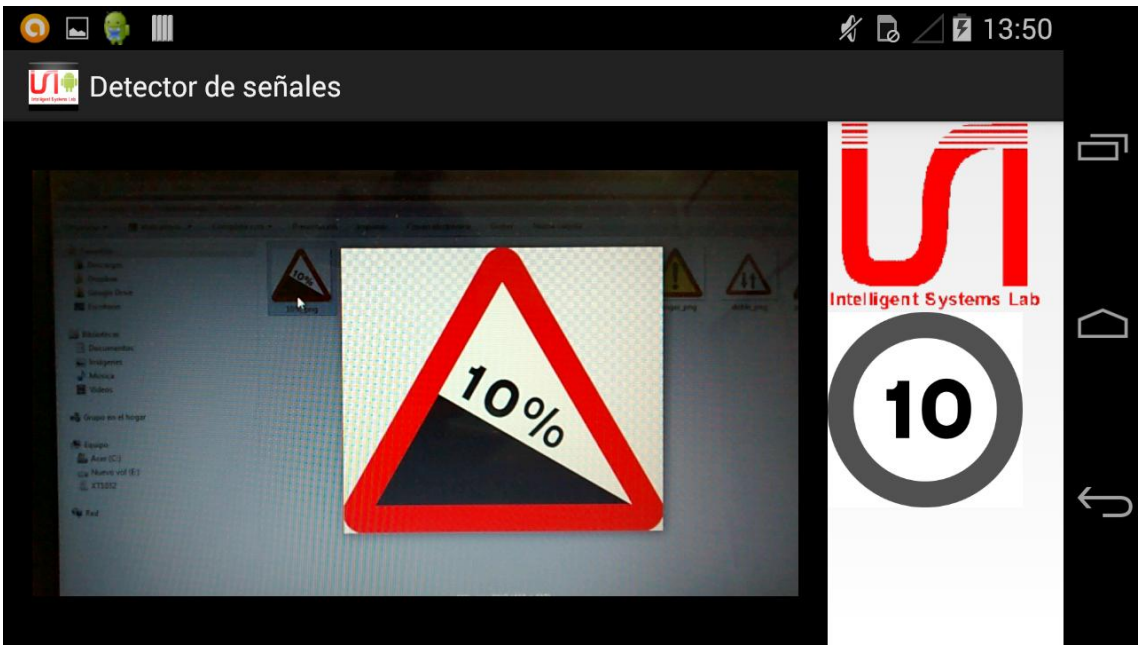

Figura 56: Problema de identificación en señal de desnivel.

En este caso se ha realizado una identificación errónea de una señal triangular de desnivel. El problema se encuentra en que la señal tiene en su interior un 10, y la clase ImageMatcher toma este como punto clave de la señal, y al realizar el emparejamiento con las señales almacenadas lo empareja con la señal de velocidad de 10 km/h.

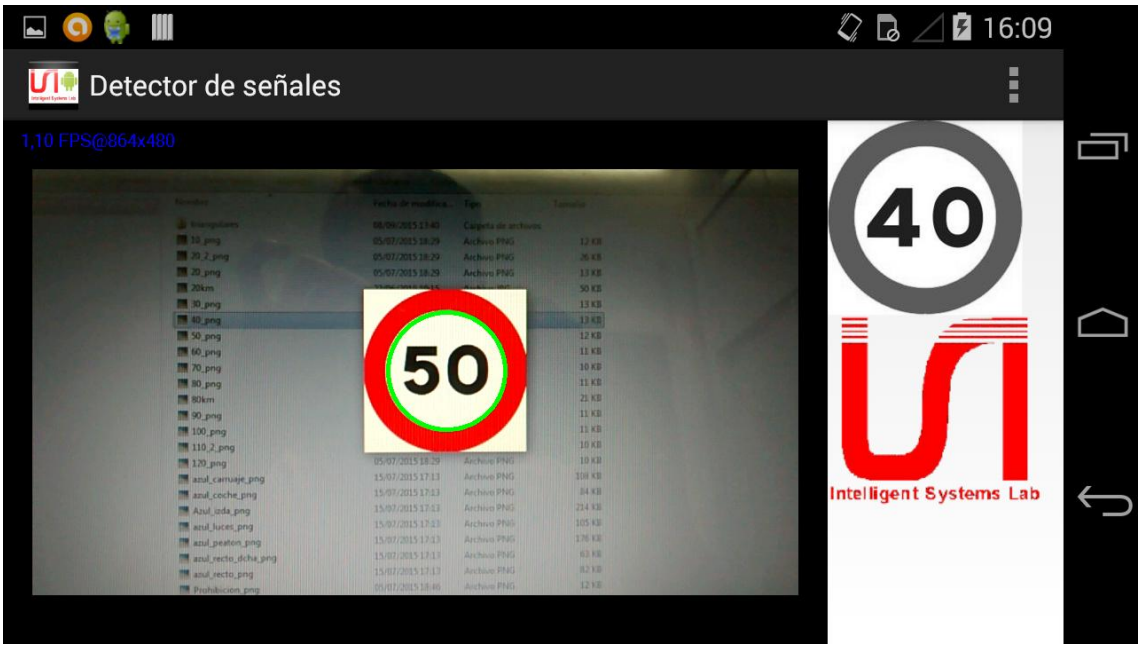

Figura 57: Problema de identificación en señal de velocidad 50 km/h.

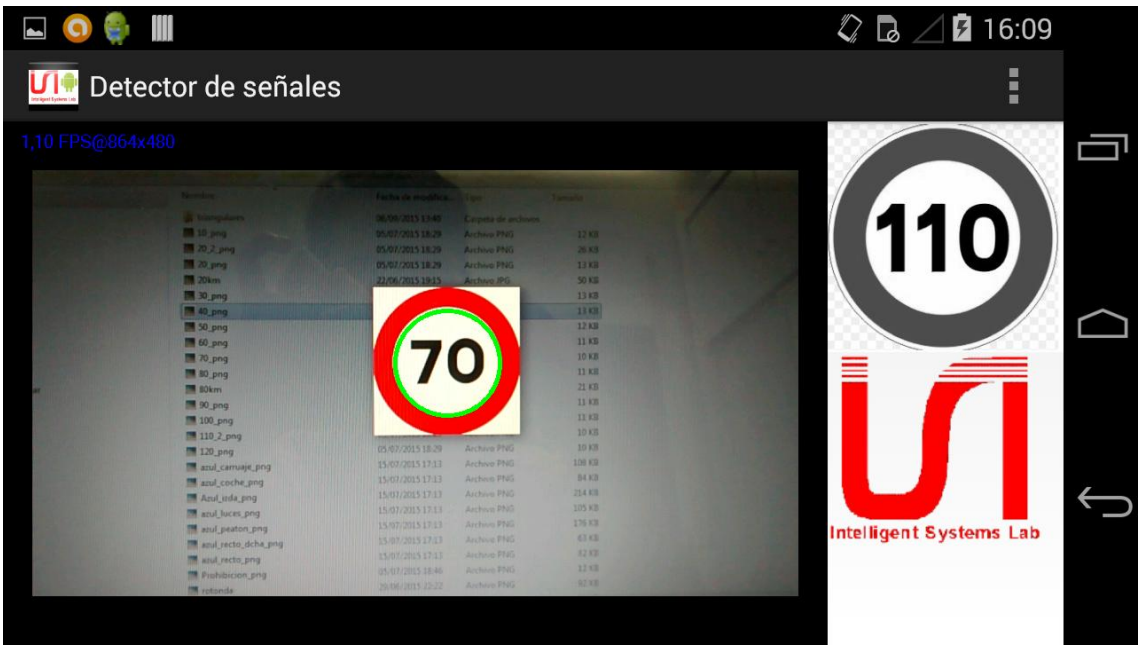

Figura 58: Problema de identificación en señal de velocidad 70 km/h.

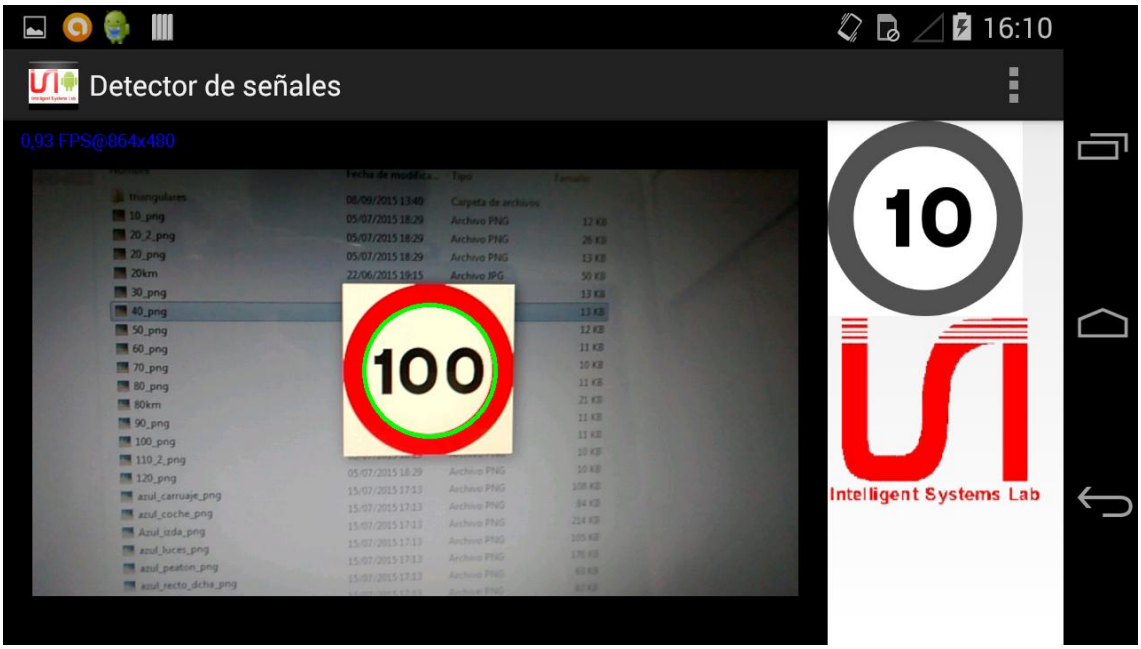

Figura 59: Problema de identificación en señal de velocidad 100 km/h.

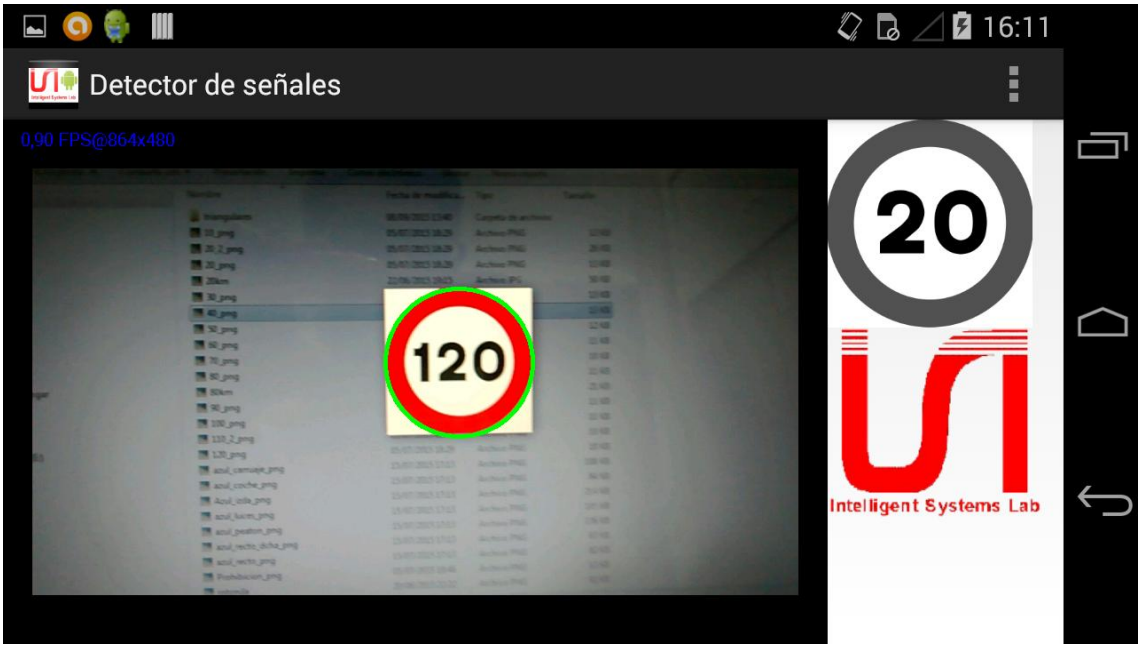

Figura 60: Problema de identificación en señal de velocidad 120 km/h.

Estos cuatro últimos casos son similares, el problema se encuentra en que el emparejador toma puntos clave muy parecidos entre todos los números, es por eso que identifica señales de velocidad que no son. En el caso de las señales de 100 y 120 km/h las confunde con las respectivas de 10 y 20 km/h por el motivo que no encuentra puntos significativos en uno de los dígitos pero si en los otros dos.

- En ocasiones se confunde una señal triangular con una cuadrada, el motivo es el mismo que en el caso anterior. Los puntos clave de ambas señales son los mismos o muy parecidos, por lo tanto la app representa la señal cuadrada azul, a pesar de que ambas están dentro de la carpeta de señales almacenadas.

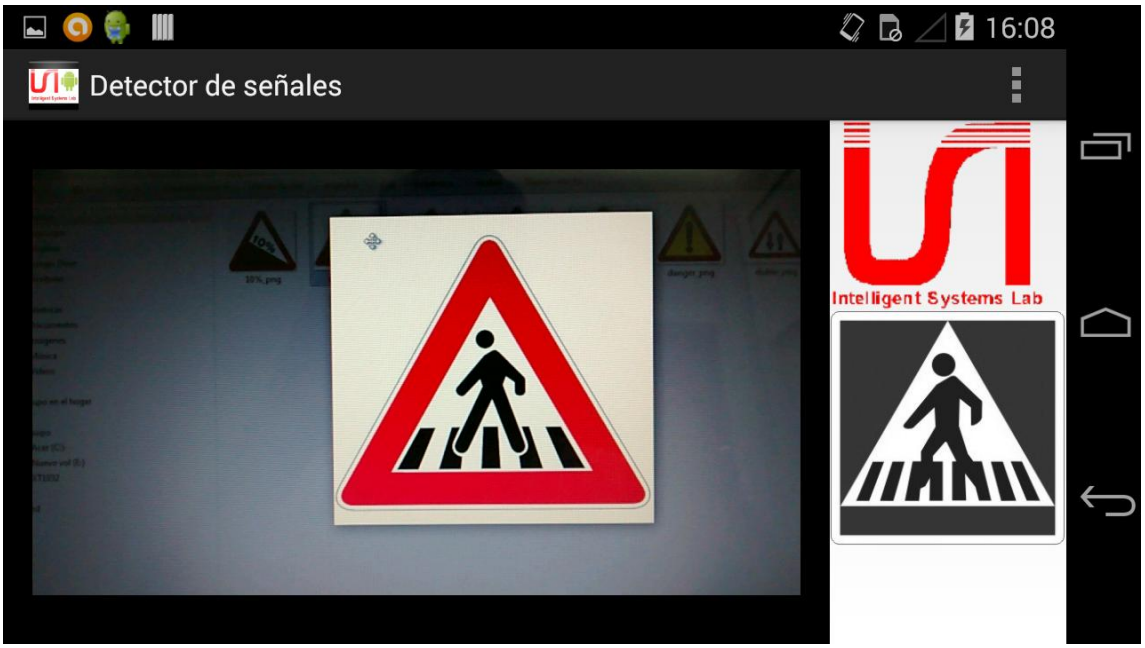

Figura 61: Problema de identificación en señal de paso de cebra.

Figura 62: Problema de identificación en señal de paso de cebra.

-

- A veces en el modo de identificación con procesado de señales circulares, se produce una doble identificación. Esta doble identificación se produce porque en primer lugar se identifica la señal después de ser procesada y modificada, y se representa en la parte derecha superior, y en ocasiones al identificarla deja de buscar círculos y entra en la búsqueda bruta, por lo tanto la representa de nuevo en la parte derecha inferior. Aquí se ve un claro ejemplo de este problema:

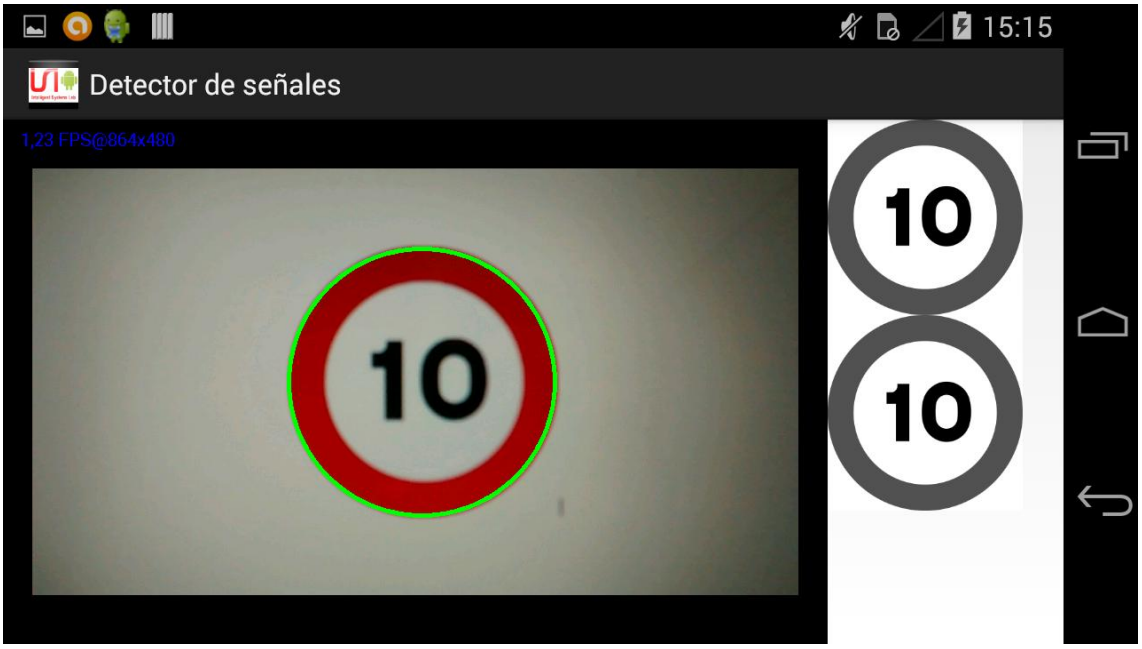

Figura 63: Problema de doble identificación de señales.

- En entorno real probado en una carretera la detección es muy defectuosa debido a la cantidad de obstáculos que la cámara recoge y factores como la velocidad y la iluminación afectan directamente a ésta identificación.
  A continuación se observa una mala identificación a causa del entorno de árboles trasero:

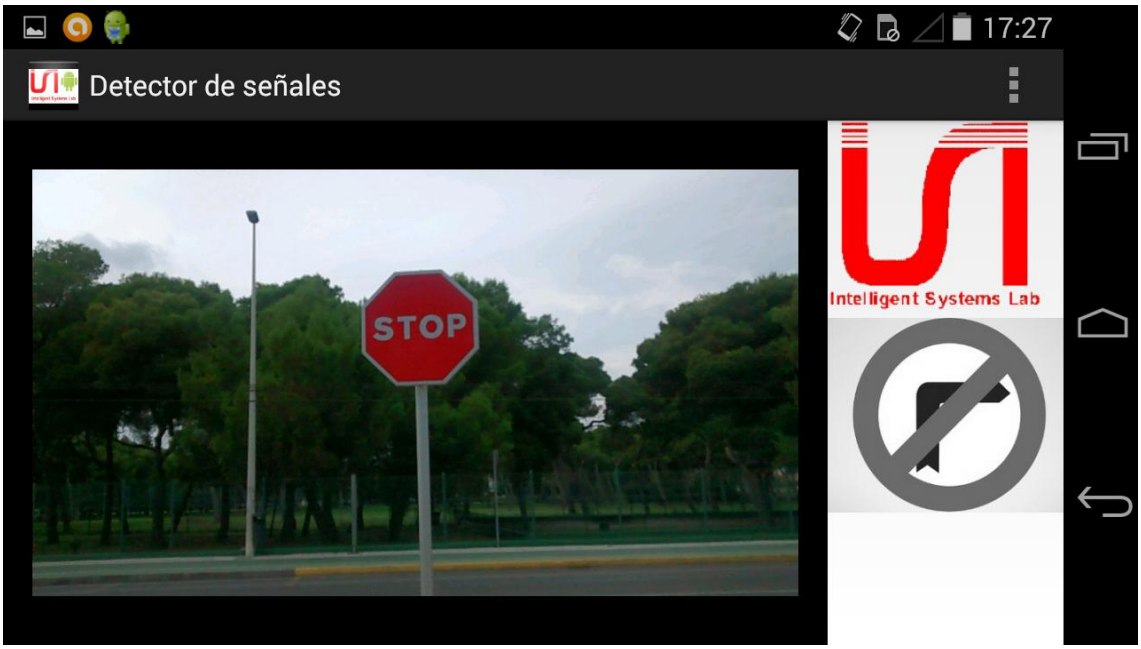

Figura 64: Problema de identificación de señal en entorno real.

- Otro problema presente en las carreteras españolas es el estado de deterioro de muchas de las señales que a causa del sol o del desgaste por el tiempo han perdido mucho color.
- A la hora de realizar el código de procesamiento de señales, en un principio se tenía la idea de procesar señales circulares y triangulares. Las circulares han sido procesadas y analizadas de manera correcta pero de cara a las triangulares no ha sido posible realizar un código que detectara y procesara estas señales, por lo tanto el código y la aplicación está pensado tan solo para procesar y analizar las señales circulares, mientras que las triangulares y otros tipos de señales se analizan sin ser procesadas.

6. Conclusiones.

6.1 Conclusiones generales.

La realización de este proyecto ha hecho que aumente considerablemente el interés hacia este tema. En un principio la visión por computador es un campo que se ha estudiado en varias asignaturas pero al involucrarse uno en un proyecto relacionado con ello, ha permitido entender muchas cosas que antes solo se veían en forma de teoría. Se expresan las conclusiones en varios aspectos:

La programación en código java no ha supuesto mucha complicación gracias a los conocimientos previos que se tienen en lenguaje C/C++ y gracias a un estudio previo realizado del lenguaje java que ayudó a entender muchas de las funciones y estructuras empleadas.

Respecto a la biblioteca OpenCV, anteriormente se había trabajado con ella pero en otro entorno, en Visual Studio. Ha sido muy similar trabajar con ella en Eclipse e incluso se ha sabido resolver muchos de los problemas que se iban encontrando gracias a la documentación que hay en internet oficial de OpenCV, además de un libro utilizado para la consulta de instalación y diferentes dudas a lo largo del proyecto.

Al estar familiarizado con el sistema operativo Android trabajar con él no ha supuesto un obstáculo. Ha resultado muy fácil y ha permitido ver que se pueden desarrollar aplicaciones de infinidad de utilidades ya que la creación de aplicaciones era algo que nunca uno se había planteado y se ha logrado de forma satisfactoria.

Personalmente los resultados de la aplicación desarrollada son bastante satisfactorios ya que permiten detectar señales a través de una cámara de teléfono móvil, en ocasiones no de manera correcta pero la mayoría de las veces sí. Es por ello que considero que el objetivo que propuesto al iniciar el proyecto de detectar señales ha sido conseguido.

6.2 Posibles mejoras.

Una de las principales mejoras en esta aplicación es elaborar más el código para que la velocidad de detección y de procesado de los fotogramas de la imagen sea mucho más rápida de la conseguida actualmente.

Por otra parte se podría desarrollar contando con la ayuda de una impresora 3D una sujeción para el vehículo la cual nos permita sujetar y enfocar de manera correcta la cámara del Smartphone sin que nos impida la visibilidad de la vía y ganando espacio frente a los tradicionales sistemas de sujeción comercializados en grandes almacenes. El problema

generalmente de éstos es que la parte de la cámara está tapada por la misma o están diseñados para otro modelo de Smartphone.

Otra posible mejora es la creación de un código que procese las señales triangulares al igual que se ha creado el mismo para las circulares, además de las cuadradas. Con esto se conseguiría detectar todo tipo de señales con un previo procesado de la información recogida por la cámara.

De éste modo no habría opción a fallo y todas las señales serían analizadas y procesadas según distintos patrones.

7. Trabajos futuros.

Un posible trabajo futuro de cara a la realización de esta aplicación a nivel universal puede ser la adaptación de la misma para otros sistemas operativos como por ejemplo iOS y el Windows Phone, y completar de éste modo que la gran mayoría de usuarios tenga acceso a la aplicación.

Respecto a otros futuros Trabajos de Fin de Grado un trabajo posible podría ser realizar el entrenamiento completo de una aplicación para poder detectar señales de otro modo.

También se pueden añadir más señales a este TFG completando así el total de las señales existentes en las carreteras de España, haciendo una identificación del 100 % de las señales.

Como se ha comentado en el apartado de posibles mejoras, se puede plantear como trabajo futuro que no solo las señales circulares sean procesadas sino todas, es decir, las cuadradas y las triangulares. Para ello sería necesario crear un código que detectara triángulos y cuadrados y posteriormente recortar las regiones de interés y analizarlas como se ha logrado conseguir en este TFG con las circulares.

Por último otro posible trabajo futuro que garantizaría el funcionamiento correcto de la aplicación sería mejorar el procesamiento de la imagen aumentando la velocidad de reconocimiento de las señales. De este modo con un entrenamiento perfeccionado la aplicación debería ser totalmente adaptable y disponible para colocar el teléfono en el salpicadero del vehículo y detectar todas las señales.

8. Presupuesto.

En este capítulo se presenta el presupuesto necesario para la realización del proyecto, este presupuesto se ha dividido en tres apartados. En primer lugar los costes del material o hardware y el coste de software utilizados durante el desarrollo del proyecto. En segundo lugar los costes del personal. Por último se ha realizado un apartado con el coste total englobando todo tipo de costes.

8.1 Coste de materiales y software utilizados.

En este apartado se incluyen todos los materiales utilizados. Se ha incluido también un apartado de software ya que se ha trabajado en conjunto con el hardware y gracias al cual se ha podido realizar las pruebas y elaborar el proyecto. Los precios están en € e incluyen los impuestos pagados en el momento de la adquisición de los equipos.

Elemento	Descripción	Coste en Euros	Unidades	Coste final
PC.	Acer Laptop Aspire AS5741-5763.	530,06 €	1	530,06
Smartphone.	Motorola Moto G.	139,75 €	1	139,75
Conjunto de Software utilizados.	OpenCV, Eclipse, Android SDK, Windows 7.	0 €	1	0
Libro estudio visión por computador.	Android Application Programming with OpenCV.	29,11 €	1	29,11
COSTE TOTAL				**698,92 €**

Figura 65: Tabla de coste de materiales y software.

8.2 Coste de personal.

En este apartado se incluye el coste del personal que corresponde al coste que tendría un ingeniero trabajando las horas necesarias para la realización del proyecto.

Para determinar el salario medio de un ingeniero hay que tener en cuenta distintos factores. En primer lugar la experiencia del mismo, la cual se establece en este caso en menos de 1 año ya que se trata de un Trabajo de Fin de Grado. Además hay que tener en cuenta que tipo de trabajo se ha desarrollado que en este caso es un trabajo de investigación. Y por último determinar donde se ha desarrollado el proyecto ya que los salarios pueden variar según el territorio donde se trabaja, en éste caso se determina España como país.

Teniendo en cuenta todos estos factores y gracias a la reciente encuesta de ingenieros industriales realizada por colegios oficiales de ingenieros industriales de Álava, Vizcaya, Guipúzcoa y Navarra se establece el sueldo en **18119** € anuales. [31]

Sueldo bruto anual	Sueldo bruto mensual	Meses de trabajo	Coste Personal (Salario ingeniero)
18119 €	1509,91 €	6 meses	9059,5 €

Figura 66: Tabla de coste de personal.

8.3 Coste total del proyecto.

En este apartado se representa mediante una tabla el coste total del desarrollo del proyecto. El coste total se establece mediante la suma de todos los costes, que en este caso son los de hardware/ software y los de personal.

Concepto	Coste
Coste hardware/software	698,92 €
Coste de personal	9059,5 €
COSTE TOTAL	**9758,42**

Figura 67: Tabla de coste TOTAL.

9. Bibliografía.

[1] Android Wikipedia [online] consultado el 15/02/2015.

https://es.wikipedia.org/wiki/Android

[2] Iautomotive [online] consultado el 15/02/2015.

http://iautomotive.co/smartphone/smartphone-operating-systems-global-market-share-2009-2014.html

[3] OpenCV Wikipedia [online] consultado el 16/02/2015.

https://es.wikipedia.org/wiki/OpenCV

[4] OpenCV [online] consultado el 16/02/2015.

http://opencv.org/

[5] OpenCV [online] consultado el 11/03/2015.

http://code.opencv.org/projects/opencv/wiki

[6] Embedded [online] consultado el 20/03/2015.

http://www.embedded.com/design/programming-languages-and-tools/4406164/Developing-OpenCV-computer-vision-apps-for-the-Android-platform

[7] [9] [11] [12] Ceautomatica [online] consultado el 21/04/2015.

http://www.ceautomatica.es/old/actividades/jornadas/XXIV/documentos/viar/61.pdf

[8] Autopista [online] consultado el 23/04/2015.

http://www.autopista.es/noticias-motor/articulo/espana-coches-circulan-46119.htm

[10] Sistemas Transporte Wikipedia [online] consultado el 27/04/2015.

https://es.wikipedia.org/wiki/Sistemas_inteligentes_de_transporte

[13] Frenomotor [online] consultado el 10/05/2015.

http://frenomotor.com/ford/sistema-limite-inteligente-velocidad-ford

[14] Automóvil Autónomo [online] consultado el 10/05/2015.

https://es.wikipedia.org/wiki/Autom%C3%B3vil_sin_conductor_de_Google

[15] Google [online] consultado el 15/05/2015.

http://www.google.com/selfdrivingcar/how/

[16] Seguridad Vial Wikipedia [online] consultado el 20/06/2015.

https://es.wikipedia.org/wiki/Seguridad_vial

[17] DGT [online] consultado el 21/06/2015.

http://www.dgt.es/Galerias/seguridad-vial/formacion-vial/cursos-para-profesores-y-directores-de-autoescuelas/doc/XIV_Curso_24_NormasYSeniales.pdf

[18] Android Wikipedia [online] consultado el 28/06/2015.

https://es.wikipedia.org/wiki/Anexo:Historial_de_versiones_de_Android

[19] OpenCV [online] consultado el 20/07/2015.

http://opencv.org/downloads.html

[20] Eclipse Wikipedia [online] consultado el 20/07/2015.

https://es.wikipedia.org/wiki/Eclipse_(software)

[21] TFG Implementación de Algoritmos de Visión por Computador en Plataforma Android. ALEJANDRO RAMOS LÓPEZ. Septiembre 2014. UC3M.

[22] Motorola [online] consultado el 22/07/2015.

http://www.motorola.com/us/smartphones/moto-g-1st-gen/moto-g.html

[23] DGT [online] consultado el 22/07/2015.

http://www.dgt.es/Galerias/prensa/2014/08/NP-campana-velocidad-agosto-2014.pdf

[24] J. Hoswse <<*Android Application Programming with OpenCV*, >> Packt Publishing, Birmingham-Mumbai. Septiembre 2013.

[25] OpenCV [online] consultado el 22/07/2015.

http://docs.opencv.org/java/org/opencv/android/CameraBridgeViewBase.html

[26] HSV Wikipedia [online] consultado el 23/07/2015.

https://es.wikipedia.org/wiki/Modelo_de_color_HSV

[27] OpenCV [online] consultado el 26/07/2015.

http://docs.opencv.org/java/

[28] OpenCV [online] consultado el 19/08/2015.

http://docs.opencv.org/modules/features2d/doc/common_interfaces_of_feature_detectors.html?highlight=featuredetector#FeatureDetector

[29] OpenCV [online] consultado el 2/09/2015.

http://docs.opencv.org/modules/features2d/doc/common_interfaces_of_descriptor_extractors.html

[30] OpenCV [online] consultado el 10/09/2015.

http://docs.opencv.org/modules/features2d/doc/common_interfaces_of_descriptor_matchers.html

[31] Coiig [online] consultado el 10/09/2015.

http://coiig.com/COIIG/index.php?option=com_content&task=view&id=86&Itemid=1222&lang=es_ES